

**UNIVERSIDAD PRIVADA ANTENOR ORREGO**  
**FACULTAD DE INGENIERÍA**  
**ESCUELA PROFESIONAL DE INGENIERÍA**  
**ELECTRÓNICA**



---

**“DISEÑO DE UNA ARQUITECTURA EMBEBIDA PARA EL  
CÁLCULO CINEMÁTICO INVERSO DE UNA EXTREMIDAD  
ROBÓTICA HEXÁPODA DE TRES GRADOS DE LIBERTAD  
MEDIANTE EL ALGORITMO CORDIC EN FPGA”**

---

**TESIS PARA OBTENER EL TÍTULO PROFESIONAL DE INGENIERO  
ELECTRÓNICO**

**LÍNEA DE INVESTIGACIÓN: APLICACIONES ROBÓTICAS**

**AUTORES:**

Br. Carlos Roger Olaya Reyes

Br. Erick Jesús Rodríguez Dávila

**ASESOR:**

Ing. Guillermo David Evangelista Adrianzén

**TRUJILLO - PERÚ**

**2018**

## **ACREDITACIONES**

Título:

**“DISEÑO DE UNA ARQUITECTURA EMBEBIDA PARA EL CÁLCULO  
CINEMÁTICO INVERSO DE UNA EXTREMIDAD ROBÓTICA HEXÁPODA DE  
TRES GRADOS DE LIBERTAD MEDIANTE EL ALGORITMO CORDIC EN  
FPGA”**

Elaborado por:

---

Br. Carlos Roger Olaya Reyes  
TESISTA

---

Br. Erick Jesús Rodríguez Dávila  
TESISTA

Aprobado por:

---

Ing. Saúl N. Linares Vértiz  
PRESIDENTE  
N° CIP 142213

---

Ing. Lenin H. Llanos León  
SECRETARIO  
N° CIP 139213

---

Ing. Oscar M. De la Cruz Rodríguez  
VOCAL  
N° CIP 85598

---

Ing. Guillermo D. Evangelista Adrianzén  
ASESOR  
N° CIP 187682

## **PRESENTACIÓN**

Señores miembros del Jurado:

De conformidad y en cumplimiento de los requisitos estipulados en el Reglamento de Grados y Títulos de la Universidad Privada Antenor Orrego y el Reglamento Interno de la Carrera Profesional de Ingeniería Electrónica para obtener el Título Profesional de Ingeniero Electrónico, ponemos a vuestra disposición el presente Trabajo de Investigación Titulado: **“DISEÑO DE UNA ARQUITECTURA EMBEBIDA PARA EL CALCULO CINEMATICO INVERSO DE UNA EXTREMIDAD ROBÓTICA HEXÁPODA DE TRES GRADOS DE LIBERTAD MEDIANTE EL ALGORITMO CORDIC EN UN FPGA”**.

Este trabajo, es el resultado de la aplicación de los conocimientos adquiridos en la formación profesional en la Universidad, excusándonos anticipadamente de los posibles errores involuntarios cometidos en su desarrollo.

Trujillo, 15 de febrero del 2018

Br. Carlos Roger Olaya Reyes  
Br. Erick Jesús Rodríguez Dávila

## DEDICATORIAS

*A Dios, por darme la oportunidad de vivir y por  
estar conmigo siempre en cada paso que doy, por  
fortalecer mi corazón e iluminar mi mente y por  
haber puesto en mi camino a aquellas personas que  
han sido mi soporte y compañía durante todo el  
período de estudio.*

*A mis padres, por ser el pilar fundamental en todo  
lo que soy, por brindarme su apoyo incondicional y  
ánimos para salir adelante.*

*A quien se ganó mi corazón y siempre me inspiro y  
motivo a continuar. Por creer en mí y acompañarme  
en todo este largo camino profesional.*

*A mis profesores, amigos, compañeros y en especial  
a mi asesor, el Ing. Guillermo Evangelista, por brindarme  
de su experiencia y dedicación en el proceso de  
realizar esta tesis.*

Erick J. Rodríguez Dávila

*A Dios, por protegerme durante todo mi camino y darme fuerzas para superar obstáculos y dificultades a lo largo de toda mi vida.*

*A mis padres y hermanos porque ellos han dado razón a mi vida, por sus consejos, su apoyo incondicional y su paciencia, todo lo que hoy soy es gracias a ellos.*

*A mi compañero, amigo y cómplice, que durante estos años de carrera ha sabido apoyarme para continuar y nunca renunciar, gracias por su amor incondicional y por su ayuda.*

*A mi asesor, el Ing. Guillermo Evangelista, por brindarme de su experiencia para el desarrollo de mi Tesis.*

Carlos R. Olaya Reyes

## **AGRADECIMIENTOS**

A Dios, por habernos dado fuerzas y ganas de seguir superarnos profesionalmente para establecer objetivos, realizar metas y cumplir responsabilidades.

A la Universidad Privada Antenor Orrego de Trujillo, por brindarnos los conocimientos necesarios para el desarrollo del Proyecto de Tesis y abrirnos el camino hacia el ámbito laboral.

A todos nuestros profesores universitarios de ingeniería electrónica, por brindarnos conocimientos en cada una de las materias tomadas para el desarrollo profesional.

A nuestros padres por enseñarnos que la mejor herencia es la educación, por sus consejos, valores, motivación y amor.

A nuestro asesor de tesis, el Ingeniero Guillermo Evangelista Adrianzén, por su confianza y apoyo en el transcurso de toda la investigación.

A todos Gracias

## **RESUMEN**

Este trabajo de investigación presenta el diseño de una arquitectura embebida para FPGA basada en CORDIC para un cálculo cinemático inverso de una extremidad robótica hexápoda de tres grados de libertad (3-DOF). Esta propuesta de diseño de arquitectura se aborda primero mediante un análisis de ecuaciones de cinemática inversa de una extremidad hexápoda de 3-DOF y como éstas son adaptadas para diseñar un esquema de arquitectura basada en operaciones de CORDIC. Después de esto, se analiza un área de trabajo de la extremidad del hexápodo de 3-DOF para obtener los requisitos de convergencia de CORDIC. Con respecto a esto, se diseñó una entidad CORDIC de punto flotante de 32 bits de alta precisión que alcanzó los requisitos de convergencia y precisión. Finalmente, se obtiene una comparación de los resultados obtenidos por la propuesta realizada y la realización de los cálculos cinemáticos en software, obteniéndose las ecuaciones de ángulos de articulación que ilustran la velocidad de procesamiento del FPGA, la precisión y los requerimientos de hardware.

## **ABSTRACT**

This research work presents a CORDIC-based FPGA realization for a three degree of free (3-DOF) hexapod leg inverse kinematics calculation. This proposal architecture design is approached first by a 3-DOF hexapod leg inverse kinematics equations analysis and how are these adaptations to design an architecture scheme based on CORDIC operations. After that, a 3-DOF hexapod leg work area is analyzed to get the CORDIC convergence requirements. Regarding to this, an iterative, high-accuracy, 32-bit floating point CORDIC entity was designed which achieved the convergence and accuracy requirements. Finally, a comparison of the results obtained by the proposal made and the realization of the kinematic calculations in software are obtained, obtaining the angles equations illustrating the precision, hardware requirements and processing speed.



## ÍNDICE

1. INTRODUCCIÓN .....	14
1.1. Realidad problemática .....	14
1.2. Delimitación del problema .....	19
1.3. Características y análisis del problema .....	20
1.4. Formulación del problema .....	20
1.5. Formulación de la Hipótesis .....	21
1.6. Objetivos de la Investigación .....	21
1.6.1. General .....	21
1.6.2. Específicos .....	21
1.7. Justificación del Estudio .....	21
1.7.1. Importancia de la Investigación .....	21
1.7.2. Viabilidad de la Investigación .....	22
1.8. Limitaciones del estudio .....	22
2. MARCO TEÓRICO .....	24
2.1. Antecedentes .....	24
2.2. Bases Teóricas .....	26
2.2.1. Robot Hexápodo .....	26
2.2.2. Cinemática de Manipuladores .....	27
2.2.2.1. Cinemática Directa .....	27
2.2.2.2. Cinemática Inversa .....	28
2.2.3. Locomoción .....	29
2.2.4. Software para cálculo matemático .....	29
2.2.5. Algoritmo CORDIC .....	30
2.2.6. Punto Flotante .....	37
2.2.7. FPGA .....	37
2.2.8. VHDL .....	39
2.2.9. Software para simulación de Lenguaje de Descripción de Hardware .....	43
2.2.10. Sistemas Embebidos .....	43
2.3. Definición de términos .....	44
2.3.1. Articulación robótica .....	44
2.3.2. Grado de libertad .....	44
2.3.3. Robot hexápodo .....	44

2.3.4.	Arquitectura embebida.....	44
2.3.5.	Concurrencia .....	44
2.3.6.	Paralelismo .....	45
2.3.7.	Área de trabajo .....	45
3.	MATERIAL Y MÉTODO.....	47
3.1.	Material .....	47
3.1.1.	Población .....	47
3.1.2.	Muestra.....	47
3.1.3.	Unidad de Análisis .....	47
3.2.	Método .....	47
3.2.1.	Nivel de Investigación .....	47
3.2.2.	Diseño de Investigación .....	47
3.2.3.	Variables de estudio y operacionalización .....	47
3.2.4.	Técnicas e instrumentos de recolección de datos .....	49
3.2.5.	Técnicas de Procesamiento de datos y análisis de datos.....	72
4.	RESULTADOS .....	79
5.	DISCUSIÓN DE RESULTADOS .....	85
6.	CONCLUSIONES .....	88
7.	RECOMENDACIONES .....	90
8.	REFERENCIAS BIBLIOGRÁFICAS .....	92
	ANEXOS .....	95

## ÍNDICE DE FIGURAS

Figura N°1.1: Mecanismos de Locomoción usados en sistemas biológicos .....	14
Figura N°1.2: a) Clasificación de robots caminantes según número de extremidades. b) Disposición de extremidades de varios animales.....	15
Figura N°1.3. a) Pata con 2 grados de libertad, b) Pata con 3 grados de libertad.....	16
Figura N°2.1: Robot Hexápodo LAURON-V del Centro de Investigación Tecnológica Alemania.....	27
Figura N°2.2: Extremidad de 3 grados de libertad.....	28
Figura N°2.3: Esquema de la Arquitectura Bit-Paralela iterativa.....	34
Figura N°2.4: Esquema de la Arquitectura Bit-Paralela Desplegada.....	35
Figura N°2.5: Esquema de la Arquitectura Bit Serie-Iterativa.....	36
Figura N°2.6: Estructura conceptual de un dispositivo FPGA.....	38
Figura N°2.7: Estructura LUT de dos entradas.....	38
Figura N°3.1: Diseño de la Investigación.....	47
Figura N°3.2: Sistema de referencia de una extremidad robótica.....	51
Figura N°3.3: Área del rango de trabajo del efector final.....	53
Figura N°3.4. Diseño de entidad CORDIC.....	58
Figura N°3.5: Diagrama de bloques para la realización de la cinemática inversa en FPGA.....	60
Figura N°3.6: Diagrama interno del bloque FB1.....	60
Figura N°3.7: Diagrama interno del bloque FB2.....	61
Figura N°3.8: Diagrama interno del bloque FB3.....	61
Figura N°3.9: Arquitectura propuesta para cinemática inversa.....	62
Figura N°3.10: Tiempo de ejecución de los ángulos de articulación.....	67
Figura N°4.1. Diseño de entidad propuesta.....	80
Figura N°4.2. Arquitectura para calcular la cinemática inversa.....	81

## ÍNDICE DE TABLAS

Tabla N°1.1: Especificaciones de hardware de robots hexápodos.....	18
Tabla N°2.1. Funciones calculadas por el algoritmo CORDIC.....	32
Tabla N°3.1. Operacionalización de la Variable Independiente.....	48
Tabla N°3.2. Operacionalización de la Variable Dependiente.....	49
Tabla N°3.3. Cuadro comparativo de actuadores.....	50
Tabla N°3.4. Análisis Rango de Convergencia CORDIC.....	54
Tabla N°3.5. $\theta_{\text{máx}}$ obtenido del nuevo rango de convergencia.....	55
Tabla N°3.6. Funciones y símbolos de los operadores CORDIC.....	56
Tabla N°3.7. Tipos de Arquitecturas para implementar CORDIC.....	57
Tabla N°3.8. Tabla de requerimiento de hardware del FPGA Spartan-3E.....	59
Tabla N°3.9. Ángulos calculados por Matlab/Caminata Trípode.....	63
Tabla N°3.10. Ángulos calculados por Matlab/Caminata Cuadrúpeda.....	64
Tabla N°3.11. Ángulos calculados por Matlab/Caminata Cuadrúpeda 4+2.....	65
Tabla N°3.12. Ángulos calculados por Matlab/Caminata Pentápoda.....	66
Tabla N°3.13. Ángulos calculados por ISim/Caminata Trípode.....	68
Tabla N°3.14. Ángulos calculados por ISim/Caminata Cuadrúpeda.....	69
Tabla N°3.15. Ángulos calculados por ISim/Caminata Cuadrúpeda 4+2.....	70
Tabla N°3.16. Ángulos calculados por ISim/Caminata Pentápoda.....	71
Tabla N°3.17 Cuadro comparativo de resolución de actuadores.....	72
Tabla N°3.18. Cuadro comparativo de los ángulos calculados por Matlab y el simulador ISim/Caminata Trípode.....	73
Tabla N°3.19. Cuadro comparativo entre ángulos calculados por Matlab y el simulador ISim/Caminata Cuadrúpeda.....	74
Tabla N°3.20. Cuadro comparativo entre ángulos calculados por Matlab y el simulador ISim/Caminata Cuadrúpeda 4+2.....	75
Tabla N°3.21. Cuadro comparativo entre ángulos calculados por Matlab y el simulador ISim/Caminata Pentápoda.....	76
Tabla N°3.22. Recursos utilizados por el dispositivo FPGA xc3s500e-4pq208.....	77
Tabla N°4.1. Operadores CORDIC.....	80
Tabla N°4.2. Error relativo porcentual según tipo de caminata.....	82
Tabla N°4.3. Recursos utilizados por arquitectura propuesta.....	83

# **CAPITULO I**

## 1. INTRODUCCIÓN

### 1.1. Realidad problemática

Según Siegwart y Nourkbakhsh (2004a), “Un robot móvil necesita *mecanismos de locomoción* que le permita desplazarse sin limitaciones. Sin embargo, hay una gran variedad de posibles formas de moverse, por la que la selección del enfoque de un robot para la locomoción es un aspecto importante del diseño del robot móvil. Existen robots de investigación que pueden caminar, saltar, correr, deslizarse, patinar, nadar, volar y rodar. La mayoría de estos mecanismos de locomoción se han inspirado en sus homólogos biológicos, como se muestra en la figura 1.1” (p.13).




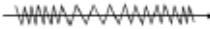

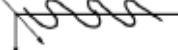

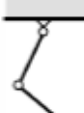




Tipo de movimiento	Resistencia al movimiento	Cinemática básica del movimiento
Fluir 	Fuerzas Hidrodinámicas	Movimiento Giratorio 
Deslizar 	Fuerzas de Fricción	Vibración Longitudinal 
Reptar 	Fuerzas de Fricción	Vibración Trasversal 
Correr 	Perdida de Energía Cinética	Movimiento oscilatorio del péndulo doble 
Saltar 	Perdida de Energía Cinética	Movimiento oscilatorio del péndulo doble 
Caminar 	Fuerzas Gravitacionales	Formando un polígono 

Figura N°1.1: Mecanismos de Locomoción usados en sistemas biológicos.

*Fuente: R. Siegwart & I.Nourbakhsh (2004). Introduction to Autonomous Mobile Robots.*

De entre estos, son los mecanismos de los robots caminantes quienes han despertado mayor interés en las últimas décadas; inspirada en algunos animales e insectos, la locomoción en estas aplicaciones es especialmente un reto en terrenos irregulares. La

locomoción de los robots caminantes permite el movimiento coordinado de los mecanismos para desplazarse libremente en los terrenos variantes (García López et al., 2012).

Debido a que los robots caminantes son inspirados biológicamente, es instructivo examinar las diferentes especies que nos rodean como los insectos y mamíferos, éstas especies se adaptan a los diferentes terrenos irregulares y pueden movilizarse sin ningún problema debido a la estructura de su anatomía. Siendo sus principales cualidades, la adaptabilidad y la maniobrabilidad en un terreno desigual. Según Holmes, Full, Koditschek y Guckenheimer (2006), los robots caminantes son clasificados según el número de extremidades como bípedos, cuadrúpedos, hexápodos y octópodos (dos, cuatro, seis y ocho patas respectivamente) como es representado en la figura 1.2a. A su vez Billah, Ahmed y Farhana (2008a) clasifica a estos robots tomando en cuenta la disposición de estas extremidades o de sus puntos de fijación respecto al cuerpo (Figura 1.2b), así como también la orientación respecto a la base.

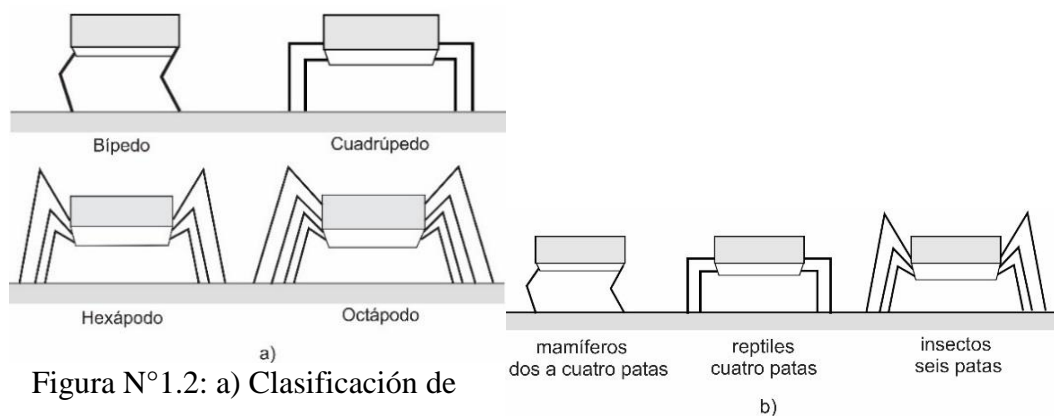


Figura N° 1.2: a) Clasificación de robots caminantes según número de extremidades. b) Disposición de extremidades de varios animales.

*Fuente: Elaboración Propia*

De estas categorizaciones de robots caminantes, los robots hexápodos presentan la mayor estabilidad estática y dinámica con la capacidad de maniobrabilidad y emular los movimientos de las demás morfologías (Evangelista, 2014).

Las extremidades en los robots caminantes se definen según el terreno en que se vaya a desenvolver y el número de grados de libertad. Se puede desarrollar una pata rígida

o curva en un robot caminante y una pata compleja para casos de movilidad variada. (Martínez y Fernández, 2003). Estas extremidades son diseñadas teniendo en cuenta sus mecanismos, debido a que estos forman parte del peso total del robot el cual debe ser soportado por estas mismas (Billah, Ahmed y Farhana, 2008b).

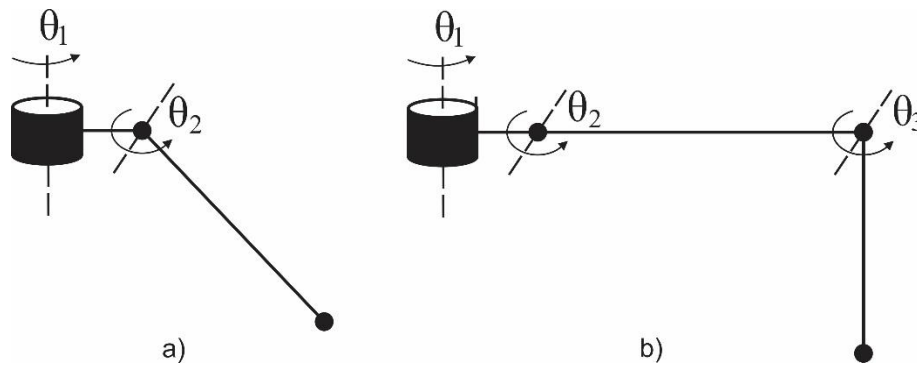


Figura N°1.3. a) Pata con 2 grados de libertad, b) Pata con 3 grados de libertad.

*Fuente: Elaboración Propia*

Dentro de los robots caminantes, un mínimo de dos grados de libertad como se ve en la figura 1.3a es generalmente requerido para mover la extremidad, elevarla y balancearla hacia adelante. Lo más común es añadir un tercer grado de libertad para conseguir maniobras más complejas, resultando una extremidad como la que se muestra en la figura 1.3b. Recientes avances en creación de robots bípedos caminantes han añadido un cuarto grado de libertad en la articulación del tobillo; en contraste, encontramos las extremidades inferiores de los humanos que tienen más de siete grados de libertad. En general, añadiendo grados de libertad a una extremidad robótica incrementa la maniobrabilidad, aumentando a su vez la gama de terrenos donde el robot puede desplazarse y la habilidad del robot para desarrollar distintos tipos de caminata. La principal desventaja de añadir articulaciones y actuadores es el aumento del consumo de energía, complejidad del control y la masa (Siegwart y Nourbakhsh, 2004b).

De esta forma, tanto la estructura del robot como la disposición de las extremidades definen la maniobrabilidad y los movimientos para los distintos tipos de caminatas que el robot puede desarrollar. Estos movimientos según su espacio y tiempo vienen siendo estudiados por la cinemática; teniendo en cuenta la posición y orientación del efector final deseado para conseguir el conjunto de ángulos que debe adoptar la



articulación para la cinemática inversa, o el de calcular la posición y orientación del efector final teniendo los ángulos de las articulaciones de la extremidad para la cinemática directa. Siendo la primera la utilizada en aplicaciones robóticas, al utilizar la posición y la orientación del efector final para obtener los parámetros geométricos que debe adoptar el robot. Para esto se aplican métodos geométricos para casos menores de tres grados de libertad (DOF, por sus siglas en inglés) y el método de la representación de Denavit - Hartenberg (DH) para casos de tres a más DOF, seleccionando adecuadamente los sistemas de coordenadas con respecto a cada grado de libertad.

Generalmente, estos parámetros son obtenidos utilizando un procesador en el robot que ejecuta los cálculos complejos. De este modo la velocidad de movimiento del robot es limitada por el hecho que estos cálculos necesitan realizarse en tiempo real (Chung, Zhang y Chen, 2015). Debido a la complejidad y tiempo requerido para el cálculo de la cinemática, es difícil encontrar una solución que pueda satisfacer los requerimientos del cálculo y el control de la locomoción en tiempo real para sistemas de múltiples grados de libertad. Las ecuaciones de la cinemática directa e inversa de una extremidad de múltiples grados de libertad envuelven un gran número de funciones trigonométricas; en un enfoque convencional, estas son calculadas empleando distintos métodos computacionales como series de Taylor, tablas de búsqueda, entre otros, los cuales son instalados en el procesador. Debido a que la performance de los controladores embebidos es limitada, este enfoque no es una solución efectiva al control en tiempo real. Es necesario encontrar un método eficiente para acelerar el control cinemático (Yili, Hanxu, Qingxuan y Guozhen, 2008).

Las ecuaciones de cinemática directa e inversa de una extremidad robótica envuelven un gran número de funciones trascendentales, estas sumadas a las características de velocidad de procesamiento, costo, nivel de integración, así como consumo de potencia; son las limitantes para seleccionar un procesador adecuado para el cálculo. Desde este enfoque, distintas propuestas han sido planteadas como se observa en la tabla 1.1.

Tabla N°1.1: Especificaciones de hardware de robots hexápodos.

Nombres de los Robots Hexápodos	LAURON V	Crixus	No Especificado	No especificado
Fecha	2014	2014	2015	2015
Autor Principal	A. Roennau	Guillermo Evangelista	Marek Zak	Antoine Cully
Procesador	Intel Core i7	ARM Cortex-M3 NXP LPC1768	Raspberry Pi	Intel Xeon E5-260
Velocidad de Procesamiento	3.0 GHz	96 MHz	1.2 GHz	2 GHz
Número de Núcleos del Procesador	4	1	4	8
Unidades de Control de Motores	9	1	1	No especificado
Consumo de Potencia (W)	100-150	60	No especificado	No especificado
Costo del Procesador (\$)	400.00*	50.00*	40.00*	70.00*

*Fuente: Elaboración Propia*

*(\*) Pololu, 2016. <https://www.pololu.com/>*

Las propuestas de la tabla anterior presentan como principal característica una unidad central de proceso que se encarga de resolver todas las rutinas requeridas por el robot, tanto como para el control, lectura de sensores, generación de trayectorias, entre otras y unidades esclavas para el control de los actuadores del robot. Enfocándose en el procesador central, se puede apreciar que estas propuestas coinciden en utilizar un procesador secuencial y dependiendo de la complejidad del control se opta por incrementar la velocidad del procesador, cantidad de núcleos y controladores de actuadores asociados. Todos estos factores influyen en el aumento de consumo de potencia, tamaño, peso y costo del robot.

Hoy en día, los procesadores secuenciales solo pueden incrementar su velocidad de procesamiento con un gran esfuerzo en desarrollo tecnológico debido a que la performance está físicamente limitada por la arquitectura. El paralelismo en los sistemas computacionales es la solución a este problema. Añadiendo unidades paralelas de procesamiento, se puede incrementar considerablemente la velocidad de procesamiento para atender problemas de cálculo complejos. Por otro lado, el paralelismo en los sistemas computacionales debe ser explotado de una forma eficiente, una mejora en la velocidad de procesamiento no puede ser conseguida solamente incrementando el número de unidades de procesamiento debido a que el tiempo necesario para la comunicación o administración de data adicional incrementa simultáneamente. Por lo tanto, es una labor importante el paralelismo de las soluciones de los problemas en robótica para adecuarlas a los dispositivos de alto nivel de paralelismo. En varios casos, se tiene que diseñar algoritmos para que el paralelismo sea factible (Henrich y Höniger, 1997a).

El procesamiento paralelo a nivel cinemático tiene dos ventajas. En primer lugar, la arquitectura del hardware del controlador refleja la arquitectura del hardware del robot, esto hace que el sistema sea más fácil de desarrollar y depurar. Segundo, estos esquemas son estáticamente extensibles, así con un apropiado algoritmo escalable, otra articulación robótica puede ser controlada añadiendo una unidad de control paralelo adicional (Henrich y Honiger, 1997b). Para que sea posible, se requiere de un dispositivo con las características adecuadas para implementar un método de cálculo concurrente que, de solución al problema cinemático inverso de una extremidad robótica, permitiendo a su vez desatender este cálculo de las funciones principales del procesador central del robot.

## **1.2. Delimitación del problema**

El trabajo de investigación se delimita al estudio, análisis y diseño de una arquitectura embebida para calcular la cinemática inversa de una extremidad robótica hexápoda de tres grados libertad con base en el algoritmo CORDIC.

### **1.3. Características y análisis del problema**

#### **Características problemáticas**

La realidad problemática estudiada tiene las siguientes características:

- Carencia de una arquitectura embebida que permita el cálculo cinemático inverso de una extremidad robótica.
- Las soluciones de cinemática inversa con dependencia de variables dificultan la adaptabilidad a un método de implementación específico en hardware.

#### **Análisis de características problemáticas**

- El cálculo cinemático en las extremidades robóticas requiere de una respuesta en tiempo real, como en el caso de los hexápodos, que por su locomoción compleja se necesita de una arquitectura embebida que permita acelerar el cálculo y brinde autonomía en su movimiento. Para los casos del hexápodo LAURON V de A. Roennau y del hexápodo con locomoción adaptativa de A. Cully, los cuales utilizan computadores avanzados con procesadores secuenciales Intel para el cálculo de la cinemática mediante software. Esto es debido, a que estos procesadores no les permiten generar arquitecturas embebidas para el cálculo mediante hardware, con lo que se mejora la velocidad de procesamiento y el cálculo en tiempo real al realizar operaciones de forma concurrente.
- Los métodos de solución cinemática deben involucran ecuaciones con funciones trascendentales y variables dependientes, esta premisa usada por G. Evangelista (2014) permite analizar la cinemática inversa de una extremidad de robot hexápodo e implementarlas en un software matemático de computador, el cual utiliza un procesador de estructura secuencial lo que causa un retardo en la ejecución de las ecuaciones, ya que dependen entre sí. Es en esa etapa, donde se presenta la dificultad para trasladar ecuaciones en una arquitectura de hardware que permita desarrollarlas en forma concurrente y de manera específica.

### **1.4. Formulación del problema**

¿Cómo adaptar los métodos de cinemática inversa a una arquitectura embebida que permita realizar el cálculo cinemático inverso de una extremidad robótica hexápoda de tres grados de libertad mediante el algoritmo CORDIC?

### **1.5. Formulación de la Hipótesis**

El diseño de una arquitectura embebida mediante el algoritmo CORDIC en FPGA permite calcular la cinemática inversa de una extremidad robótica hexápoda de tres grados de libertad.

### **1.6. Objetivos de la Investigación**

#### **1.6.1. General**

Diseñar una arquitectura embebida en un FPGA que permita calcular la cinemática inversa de una extremidad robótica hexápoda de tres grados de libertad con base al algoritmo CORDIC.

#### **1.6.2. Específicos**

- Adaptar la solución de la cinemática inversa de una extremidad robótica hexápoda de tres grados de libertad para su desarrollo con base al algoritmo CORDIC.
- Estudiar y analizar los modos, limitaciones y parámetros de diseño del algoritmo CORDIC.
- Diseñar una entidad implementable en FPGA para el algoritmo CORDIC en VHDL.
- Diseñar una arquitectura embebida en FPGA empleando las entidades CORDIC diseñadas para la solución de la cinemática inversa.

### **1.7. Justificación del Estudio**

#### **1.7.1. Importancia de la Investigación**

El presente trabajo de investigación conseguirá desarrollar una arquitectura embebida que permita calcular la cinemática inversa de una extremidad robótica hexápoda de tres grados de libertad de forma concurrente. Esto representa un aporte a los sistemas computacionales, brindando una nueva alternativa para mejorar el desarrollo de procesadores especializados de robótica en tiempo real.

### **1.7.2. Viabilidad de la Investigación**

Para el desarrollo de esta investigación se cuenta con los recursos bibliográficos, plataformas de desarrollo, simulación y análisis, además del asesoramiento en el ámbito de sistemas computacionales y robótica por parte de nuestro asesor. Se cuenta con los recursos de hardware necesarios para poder realizar la implementación de la arquitectura planteada, así como de una plataforma hexápoda realizada en una investigación pasada por la Escuela Profesional de Ingeniería Electrónica de la Universidad Privada Antenor Orrego.

- **Viabilidad técnica**

Para la realización de esta investigación se cuenta con los recursos técnicos necesarios como: tiempo, conocimientos, información científica y un asesor que nos ayudará a resolver dudas durante el desarrollo.

- **Viabilidad económica**

La presente investigación cuenta con un presupuesto para la adquisición de equipos y materiales necesarios, solventado por los mismos autores.

- **Viabilidad social**

Esta investigación puede repercutir en futuras investigaciones sobre aplicaciones robóticas autónomas en beneficio de mejorar la calidad de vida humana.

### **1.8. Limitaciones del estudio**

El presente proyecto de investigación no se encuentra limitado en ningún aspecto, ya que se cuenta con los recursos necesarios para poder desarrollar la arquitectura embebida en FPGA para el cálculo de la cinemática inversa de una extremidad robótica hexápoda de tres grados de libertad con base en el algoritmo CORDIC.

# **CAPITULO II**

## 2. MARCO TEÓRICO

### 2.1. Antecedentes

- **“Fully pipelined CORDIC-based inverse kinematic FPGA design for biped robots”**

*Autor:* Rih-Lung Chung, Yi-Qin Zhang y Shih-Lun Chen

*Institución:* Chung Yuan Christian University, Taiwan

*Año:* 2015

***Conclusiones:***

Este artículo científico, propone un algoritmo orientado a hardware de baja complejidad y alta precisión para robots bípedos. Mediante el uso de técnicas de distribución de hardware y de pipeline. Su propuesta tiene beneficios en bajo costo, alta precisión y alto rendimiento.

***Aporte al Trabajo de Investigación:***

El aporte principal del artículo son las técnicas de distribución de hardware, así como la de pipeline utilizada, método que fue utilizado en el diseño de la arquitectura embebida.

- **“Design and modeling of a mobile research platform based on hexapod robot with embedded system and interactive control”**

*Autor:* Evangelista Adrianzén, Guillermo

*Institución:* Universidad Privada Antenor Orrego, Perú

*Año:* 2014

***Conclusiones:***

Este trabajo presenta el desarrollo de una plataforma de investigación robótica hexápoda para uso académico. El desarrollo hace hincapié en el carácter procedimental y se ha definido etapas: diseño mecánico, sistema de modelado, incorporados del sistema e interfaz de control. El diseño mecánico: modelo preliminar, la simulación de movimiento, físicas de materiales y fabricación. El sistema de modelado: cinemática, la dinámica y la locomoción. El diseño incorporado: unidad de proceso y servo-controlador; mientras que la interfaz de control implementa todos los estudios. Por último, los procedimientos validados



demuestran que es una plataforma de investigación versátil, que permite validar las técnicas y estudios, y también fortalece el proceso de enseñanza y aprendizaje.

***Aporte al Trabajo de Investigación:***

El principal aporte ha sido el sistema de modelado: cinemática, la dinámica y la locomoción, cuyas ecuaciones de la cinemática inversa, así como la simulación de movimiento serán utilizadas para el desarrollo de esta investigación.

- **“Implementación de arquitecturas para el cálculo de funciones trascendentales empleando el algoritmo CORDIC en un FPGA”**

***Autor:*** Agurto Rios, Carla Paola

***Institución:*** Pontificia Universidad Católica del Perú, Perú.

***Año:*** 2006

***Conclusiones:***

Se diseñaron distintos tipos de arquitectura para implementar el algoritmo CORDIC en un sistema CORDIC, las cuales son: Iterativa, Serial y Pipeline. Son parametrizables el tamaño de bus de datos (12,16,24 y 32 bits) y el modo de operación del algoritmo. De esta manera se puede escoger la arquitectura más óptima según la función que se necesite implementar. Este sistema descrito se implementó en su totalidad usando el lenguaje de descripción de hardware VHDL y se simuló usando el programa QUARTUS II 5.0 de ALTERA CORPORATION.

***Aporte al Trabajo de Investigación:***

Este presente trabajo de tesis de pregrado, aporta con los tipos de arquitecturas para la implementación del algoritmo de CORDIC, así como pruebas de implementación de arquitecturas con diferentes tamaños de bus de datos y precisión.

- **“Expanding the Range of Convergence of the CORDIC Algorithm”**

***Autor:*** Xiaobo Hu, Ronald G. Harber y Steven C. Bass

***Institución:*** Purdue University, USA

***Año:*** 1991

### ***Conclusiones:***

Proponen que el algoritmo CORDIC, tiene limitaciones en el tamaño de la entrada de cantidades que impide que el algoritmo sea ampliamente útil cuando se implementa como un procesador aritmético de propósito especial o aplicación similar. Las modificaciones discutidas en este documento permiten que los rangos de convergencia se expandan como lo exigen las especificaciones del sistema, manteniendo al mismo la simplicidad del algoritmo original. Proponen métodos para expandir el rango de convergencia para el algoritmo CORDIC que no requiere de ningún cálculo excesivo adicional. Haciendo así este trabajo muy favorable para una implementación en hardware. Además, el número de iteraciones adicionales que se tienen en los algoritmos de CORDIC modificados con significativamente menores que los planteados en otros artículos.

### ***Aporte al Trabajo de Investigación:***

Este artículo contribuye con técnicas de expansión del rango de convergencia para el cálculo de algunas funciones trascendentales aplicando el algoritmo de CORDIC, generando algoritmos de CORDIC modificados con menores iteraciones lo que permite la implementación en hardware.

## **2.2. Bases Teóricas**

### **2.2.1. Robot Hexápodo**

Un robot hexápodo es un vehículo mecánico con seis patas. Es establemente estático utilizando de tres a más patas y la capacidad de poder desplazarse con diferentes tipos de caminatas, según la cantidad de patas que utilice. Estos robots caminantes son adecuados para diferentes aplicaciones en diversidad de terrenos, ya que se caracterizan por tener movimientos omnidireccionales, geometría variable, buena estabilidad y locomoción tolerante a fallos (Tedeschi y Carbone, 2014).

Uno de los robots hexápodos más resaltantes desarrollados en los últimos años es el LAURON V (figura 2.1), el cual es el resultado de 10 años de investigación, el cual fue inspirado biológicamente en un insecto palo. La extremidad de este robot posee cuatro articulaciones, un sensor de fuerza de tres ejes y cada motor con un

sensor de corriente que detecta fuerzas opuestas a su movimiento (Roennau, Heppner, Nowicki y Dillman, 2014).



Figura N°2.1: Robot Hexápodo LAURON-V del Centro de Investigación Tecnológica. Alemania.

*Fuente: “LAURON V: A Versatile Six-Legged Walking Robot with Advanced Maneuverability “(Roennau, Heppner, Nowicki and Dillman,, 2014).*

### **2.2.2. Cinemática de Manipuladores**

Craig (2006), menciona que: “La cinemática es la ciencia del movimiento que trata el tema sin considerar las fuerzas que lo ocasionan. Dentro de esta se estudian la posición, velocidad, aceleración y todas las demás derivadas de alto orden de las variables de posición (con respecto al tiempo o a cualquier otra variable). En consecuencia, el estudio de la cinemática de manipuladores se refiere a todas las propiedades geométricas y basadas en el tiempo del movimiento. Las relaciones entre estos movimientos, las fuerzas y los momentos de torsión que los ocasionan constituyen el problema de la dinámica”.

#### **2.2.2.1. Cinemática Directa**

Consiste en determinar la posición y la orientación del extremo final de un robot, con respecto a un sistema de coordenadas inicial como referencia, siendo datos

conocidos los valores de los ángulos de articulación y los parámetros geométricos del robot.

Basándonos en el análisis de las posiciones en el espacio del efector final realizado por Evangelista (2014), tenemos en cuenta las siguientes ecuaciones (2.01, 2.02 y 2.03) para la cinemática directa.

$$x_e = \cos \theta_1 (L_3 \cos(\theta_2 + \theta_3) + L_2 \cos \theta_2 + L_1) \quad (2.01)$$

$$y_e = \sin \theta_1 (L_3 \cos(\theta_2 + \theta_3) + L_2 \cos \theta_2 + L_1) \quad (2.02)$$

$$z_e = L_3 \sin(\theta_2 + \theta_3) + L_2 \sin \theta_2 \quad (2.03)$$

### 2.2.2.2. Cinemática Inversa

Al contrario de la cinemática directa, la inversa determina los valores de los ángulos de articulación a partir de la posición del efector final y los parámetros geométricos del robot. Considerando la estructura general de una extremidad robótica hexápoda de tres grados de libertad planteada por Evangelista (2014), representada en la figura 2.2, la cual se dispone en estudio y se obtienen las ecuaciones de los tres ángulos de articulación.

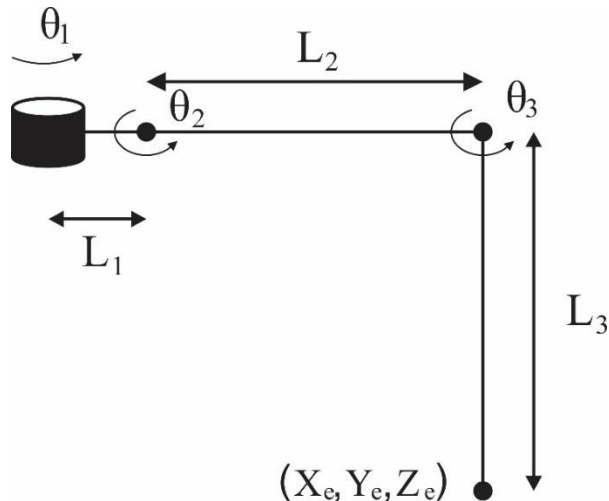


Figura N°2.2: Extremidad de 3 grados de libertad.

*Fuente: Elaboración Propia.*

Según el trabajo de investigación realizado por Evangelista (2014), como resultado del análisis cinemático de una extremidad robótica hexápoda, se obtienen las

ecuaciones (2.1), (2.2) y (2.3) de los ángulos de articulación de la cinemática inversa de una extremidad de tres grados de libertad para un robot hexápodo.

$$\theta_1 = \tan^{-1} \left( \frac{y_e}{x_e} \right) \quad (2.1)$$

$$\theta_2 = \operatorname{asin} \left( \frac{z_e}{\sqrt{x_e^2 + y_e^2 + z_e^2 + L_1^2 - 2L_1\sqrt{x_e^2 + y_e^2}}} \right) - \operatorname{atan} \left( \frac{L_3 \sin \theta_3}{L_2 + L_3 \cos \theta_3} \right) \quad (2.2)$$

$$\theta_3 = \operatorname{acos} \left( \frac{x_e^2 + y_e^2 + z_e^2 + L_1^2 - L_2^2 - L_3^2 - 2L_1\sqrt{x_e^2 + y_e^2}}{2L_2L_3} \right) \quad (2.3)$$

### 2.2.3. Locomoción

Cuaya (2007), indica que para un robot polípedo (múltiples articulaciones) la locomoción es el resultado de un movimiento coordinado de sus patas. Este movimiento se define por cierto paso que refleja determinadas especificaciones tales como velocidad, dirección, etc. Distingue dos fases claramente que contribuyen al movimiento de cada pata del robot: la *fase de soporte* y la *fase de transferencia*. Durante la fase de soporte o apoyo, cada pata debe ser capaz de ejercer cierta fuerza sobre la superficie en la que se encuentra el robot, de forma ordenada para proporcionar la fuerza necesaria al cuerpo del robot para así permitirle moverse según una trayectoria predeterminada. Después, durante la fase de transferencia o transición, la pata debe desplazarse de forma ordenada hacia el siguiente punto de soporte para reiniciar la secuencia de movimiento.

### 2.2.4. Software para cálculo matemático

Actualmente existen muchos softwares que permiten el cálculo matemático los cuales contienen distintas herramientas que permitirán su resolución de manera rápida y precisa. Algunos de estos se presentan a continuación:

- Matlab
- Calc 3d Prof
- LinCalc
- GNU Octave

- Scilab

#### **2.2.4.1. Matlab (Matrix Laboratory)**

MATLAB es un programa que millones de ingenieros y científicos de todo el planeta utilizan para analizar y diseñar sistemas y productos que transforman nuestro mundo. Esta plataforma está optimizada para resolver problemas de ingeniería y científicos de todo el mundo lo utilizan para distintas aplicaciones (Matlab, 2017). En todas las aplicaciones se utiliza para computación numérica y visualización de datos, basado en un software de matrices para el análisis de sistemas de ecuaciones. De esta forma permite resolver complicados problemas numéricos sin necesidad de escribir un programa, en donde se pueden encontrar representados cálculos matemáticos y la visualización gráfica de los mismos.

Asimismo, destaca por su bajo costo y por su versión educativa para estudiantes de todo el mundo.

Por estas características, es que se elige usar el software Matlab para realizar el cálculo cinemático inverso cuyos resultados se tomarán como punto de comparación.

#### **2.2.5. Algoritmo CORDIC**

El algoritmo CORDIC (Coordinate Rotation DIgital Computer) es un computador digital de propósito específico diseñado originalmente para aplicaciones aeroespaciales en tiempo real. Este utiliza una única técnica computacional que se adecua especialmente para resolver la relación trigonométrica envuelta en rotaciones coordinadas y conversiones de coordenadas rectangulares a polares. La formulación iterativa de este algoritmo fue descrita por primera vez en 1959 por Jack. E. Volder.

El concepto principal de este algoritmo está basado en un simple fundamento de la geometría plana. El algoritmo CORDIC provee un eficiente método para rotar vectores en un plano empleando simples sumas, restas y desplazamientos binarios

para estimar funciones básicas como operaciones trigonométricas, multiplicación, división y muchas otras operaciones como funciones logarítmicas, exponenciales y raíz cuadrada (Arora, Chauhan y Bagga, 2012).

En 1971, Walther encontró que las iteraciones del algoritmo CORDIC pueden ser modificadas para calcular funciones hiperbólicas y reformuló el algoritmo en una forma generalizada la cual se adecua para realizar rotaciones en los sistemas de coordenadas circular, hiperbólico y lineal. La formulación general del algoritmo CORDIC de la siguiente manera:

$$x_{i+1} = x_i - m \cdot d_i \cdot 2^{-i} \cdot y_i \quad (2.4)$$

$$y_{i+1} = y_i + d_i \cdot 2^{-i} \cdot x_i \quad (2.5)$$

$$z_{i+1} = z_i - d_i \cdot \alpha_i \quad (2.6)$$

Donde:

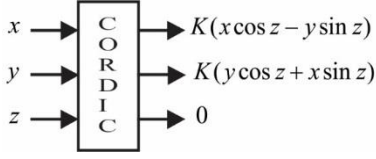
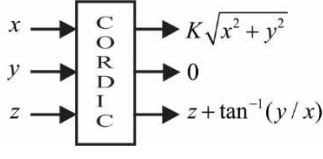
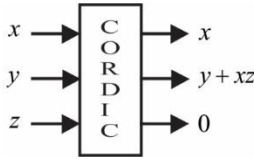
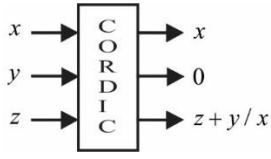
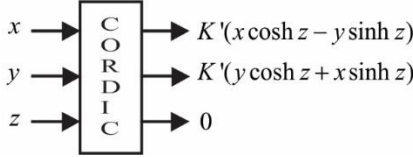
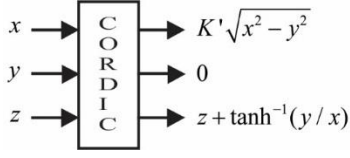
$$d_i = \begin{cases} \text{sign}(z_i), & \text{modo rotación} \\ -\text{sign}(y_i), & \text{modo vectorización} \end{cases}$$

Para  $m = 1, 0$  o  $-1$ , y  $\alpha_i = \tan^{-1}(2^{-i})$ ,  $2^{-i}$  o  $\tanh^{-1}(2^{-i})$ , el algoritmo dado por las ecuaciones (2.4), (2.5) y (2.6) son utilizables en sistemas de coordenadas circulares, lineales o hiperbólicas, respectivamente (Meher, Valls, Juang, Sridharan y Maharatna, 2009).

### 2.2.5.1. Modos de operación del algoritmo CORDIC

Las iteraciones del algoritmo CORDIC pueden ser realizadas en dos modos de operación denominados modo rotación y vectorización, los cuales básicamente difieren en la dirección de las rotaciones. Se puede calcular un número de funciones trascendentales partiendo de la formulación general como se muestra en la tabla 2.1 (Dawid y Meyr, 1999).

Tabla N°2.1. Funciones calculadas por el algoritmo CORDIC.

Modo	Rotación: $d_i = \text{sign}(z_i)$ , $z_i \rightarrow 0$	Vectorización: $d_i = -\text{sign}(y_i)$ , $y_i \rightarrow 0$
Circular  $m = 1$ $\alpha_i = \tan^{-1} 2^{-i}$	 <p>Para <math>\cos</math> y <math>\sin</math>, <math>x = 1/K</math>, <math>y = 0</math></p> <p><math>\tan z = \sin z / \cos z</math></p>	 <p>Para <math>\tan^{-1}</math>, <math>x = 1</math>, <math>z = 0</math></p> <p><math>\cos^{-1} w = \tan^{-1}[\sqrt{1 - w^2}/w]</math></p> <p><math>\sin^{-1} w = \tan^{-1}[w/\sqrt{1 - w^2}]</math></p>
Lineal  $m = 0$ $\alpha_i = 2^{-i}$	 <p>Para multiplicación, <math>y = 0</math></p>	 <p>Para división, <math>z = 0</math></p>
Hiperbólico  $m = -1$ $\alpha_i = \tanh^{-1} 2^{-i}$	 <p>Para <math>\cosh</math> y <math>\sinh</math>, <math>x = 1/K'</math>, <math>y = 0</math></p> <p><math>\tanh z = \sinh z / \cosh z</math></p> <p><math>\exp(z) = \sinh z + \cosh z</math></p> <p><math>w^t = \exp(t \ln w)</math></p>	 <p>Para <math>\tanh^{-1}</math>, <math>x = 1</math>, <math>z = 0</math></p> <p><math>\ln w = 2 \tanh^{-1}  (w - 1)/(w + 1) </math></p> <p><math>\sqrt{w} = \sqrt{(w + 1/4)^2 - (w - 1/4)^2}</math></p> <p><math>\cosh^{-1} w = \ln(w + \sqrt{1 - w^2})</math></p> <p><math>\sinh^{-1} w = \ln(w + \sqrt{1 + w^2})</math></p>

Fuente: Elaboración Propia

De la tabla anterior se considera el factor de escala  $K$ , que viene dado por la ecuación (2.7).

$$K = \prod_{i=0}^n (1 + 2^{-2i})^{-1/2} \quad (2.7)$$

Además, en la ejecución de las iteraciones para  $m = -1$ , los pasos 4,13,40, 121,...,  $j$ ,  $3j+1$ ,... deben repetirse. Estas repeticiones son incorporadas en la constante  $K'$



### 2.2.5.2. Implementación en Hardware del algoritmo CORDIC

El algoritmo CORDIC puede ser implementado en hardware mediante tres arquitecturas, según (Schweers, 2012), las clasifica de la siguiente manera:

- **Arquitectura Bit-Paralela Iterativa**

Se denomina paralela a la forma en que se opera con los componentes  $X$ ,  $Y$  y  $Z$ . En la arquitectura cada etapa del algoritmo consiste en un registro para almacenar la salida, una unidad de desplazamiento y un sumador algebraico. Para iniciar el cálculo, los valores iniciales para  $x_0$ ,  $y_0$  y  $z_0$  ingresan en forma paralela a los registros a través de un multiplexor. El bit más significativo del componente  $Z$  o  $Y$  en cada paso de iteración determina la operación a efectuar por el sumador algebraico, en modo rotación o vectorización respectivamente. Las señales correspondientes a los componentes  $X$  e  $Y$  son desplazadas y luego restadas o sumadas a las señales sin desplazar, correspondientes al componente opuesto. El componente  $Z$  combina los valores almacenados en el registro con valores que obtiene de una tabla de búsqueda (Lookup Table, LUT) de arcotangentes pre-calculadas, con una cantidad de entradas proporcional a la cantidad de iteraciones.

Se requiere de un controlador que puede ser implementado como una máquina de estados para controlar los multiplexores, la cantidad de desplazamientos y el direccionamiento de las constantes pre-calculadas. Esta presentación tiene como ventaja el uso eficiente de hardware, debido a que los recursos (registros, multiplexores, unidades de desplazamiento y sumadores) son reutilizados en cada iteración. En la figura 2.3 se muestra el esquema correspondiente a la arquitectura descrita. La señal “Modo” especifica el modo de operación (Rotación o Vectorización). Se considera además en el esquema que las componentes tienen un ancho de “ $m$ ” bits. Sin embargo, se puede ampliar el ancho en bits de los buses internos para obtener una mejor exactitud.

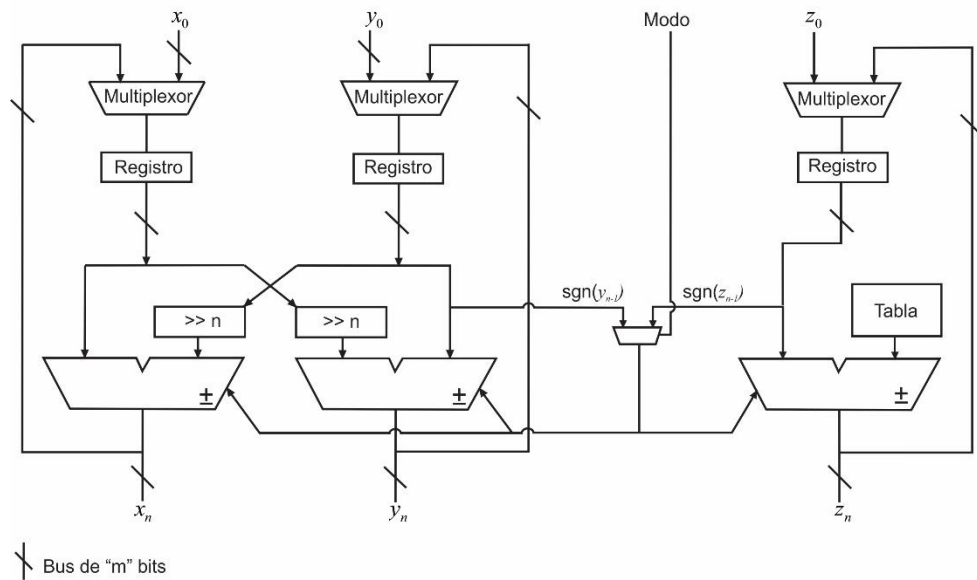


Figura N°2.3: Esquema de la Arquitectura Bit-Paralela Iterativa.

*Fuente: Elaboración Propia.*

#### ▪ **Arquitectura Bit-Paralela Desplegada**

En lugar de almacenar el resultado de cada paso de iteración en registros y volver a utilizar los mismos recursos, el diseño puede desplegarse como muestra en la figura 2.4. El diseño se separa en etapas correspondientes a cada iteración. Cada etapa está compuesta por los mismos componentes, dos unidades de desplazamiento y dos sumadores algebraicos. Por lo tanto, la salida de una etapa corresponde a la entrada de la siguiente etapa. Los valores iniciales para  $x_0$ ,  $y_0$  y  $z_0$  se ingresan en paralelo a la primera etapa.

Esta arquitectura introduce dos ventajas importantes, la primera es que las unidades de desplazamiento, así como las constantes correspondientes a cada iteración pueden ser cableadas. La segunda ventaja radica en que el circuito es puramente combinatorio, y por lo tanto no se necesita de una unidad de control, simplificando enormemente el diseño. La desventaja que presenta esta arquitectura es la enorme cantidad de espacio que requiere su implementación. En el esquema se considera un ancho de palabra de “m” bits. La señal “Modo” indica el modo de operación al igual que en el caso iterativo.

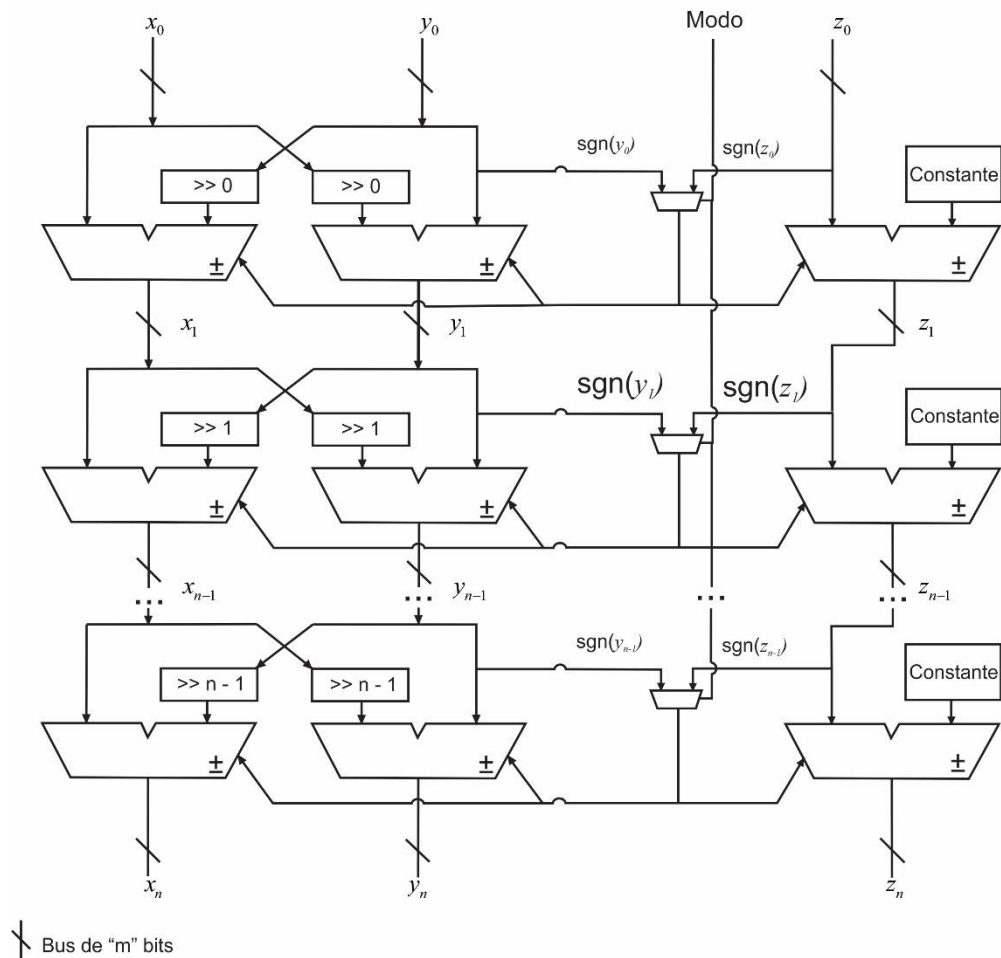


Figura N°2.4: Esquema de la Arquitectura Bit-Paralela Desplegada.

Fuente: Elaboración Propia.

#### ▪ Arquitectura Bit-Serie Iterativa

En este diseño en serie, se procesa un bit a la vez, con lo cual las interconexiones reducen el ancho de bit.

La arquitectura resultante se muestra en la figura 2.5. Cada etapa del algoritmo consiste en un multiplexor, un registro de desplazamiento y un sumador algebraico bit-serie. El sumador algebraico se implementa como un sumador completo, en el que una resta se lleva a cabo sumando el complemento a dos del valor a restar. La operación a efectuar por el sumador algebraico se obtiene del bit de signo del componente  $z$  para el modo rotación. Para el modo vectorización, el bit de signo se obtiene del componente  $y$ .

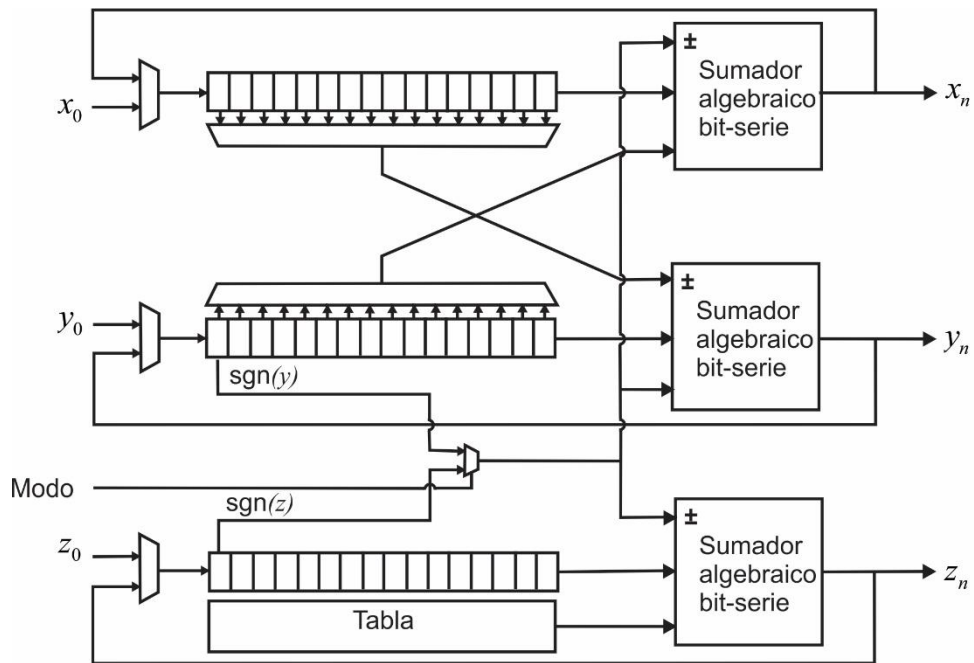


Figura N° 2.5: Esquema de la Arquitectura Bit Serie-Iterativa.

Fuente: Elaboración propia.

La operación de desplazamiento (multiplicación por  $2^{-i}$ ) se lleva a cabo leyendo el bit  $i-1$  considerando a partir del extremo derecho del registro de desplazamiento. Se puede usar un multiplexor para cambiar la posición de acuerdo a la iteración actual. Los valores iniciales para  $x_0$ ,  $y_0$  y  $z_0$  ingresan al registro de desplazamiento por el extremo izquierdo en el esquema. Cuando el primer bit que fue ingresado al registro es procesado por el sumador algebraico, el multiplexor permite nuevamente el ingreso de los bits sumados al registro de desplazamiento. Finalmente, cuando se han completado todas las iteraciones, los multiplexores permiten el ingreso de un nuevo valor al registro de desplazamiento y el valor calculado se obtiene en la salida. Al mismo tiempo, la carga y lectura del registro de desplazamiento puede efectuarse en paralelo para simplificar el diseño. La desventaja que presenta esta arquitectura con respecto a las que procesan los vectores de entrada en forma paralela es que se introduce un retardo proporcional al ancho de la palabra en cada etapa, ocasionado por los desplazamientos. Por otra parte, se requiere de hardware de control más complejo. La ventaja que presenta esta arquitectura es que los buses de interconexión entre los componentes son del ancho de un bit y el

registro de desplazamiento está integrado con el registro intermedio, minimizando espacio al momento de su implementación.

#### **2.2.6. Punto Flotante**

Punto flotante es un método para representar a los números reales, denominado así debido a que, en los procesadores, el punto binario puede *flotar* relativamente entre los dígitos significantes del número. Esta posición es indicada de forma separada en la representación del número, de esta forma el formato Punto Flotante puede ser pensado como la representación computacional de la notación científica.

A través de los años, distintas representaciones del formato *punto flotante* han sido utilizadas en los computadores; sin embargo, en los últimos diez años es el estándar IEEE-754, para simple y doble precisión del punto flotante, el más utilizado.

Para las operaciones aritméticas en números de coma flotante como suma, substracción, multiplicación y división se realizan con algoritmos similares a los utilizados en magnitudes enteras con signo.

#### **2.2.7. FPGA**

Chu (2008), define un FPGA (Field Programmable Gate Array) de la siguiente manera: es un dispositivo lógico que contiene una estructura bidimensional de matrices de celdas genéricas lógicas y compuertas programables. La estructura conceptual de un dispositivo FPGA se muestra en la figura 2.6. Una celda lógica puede ser configurada para realizar una función simple, y una compuerta programable puede ser personalizada para proporcionar interconexiones entre las celdas lógicas. Se puede implementar un diseño personalizado especificando la función de cada celda lógica y establecer selectivamente la conexión de cada compuerta programable. Una vez completado el diseño y la síntesis, podemos utilizar un cable adaptador simple para descargar la configuración de la celda y la compuerta deseada al dispositivo FPGA.

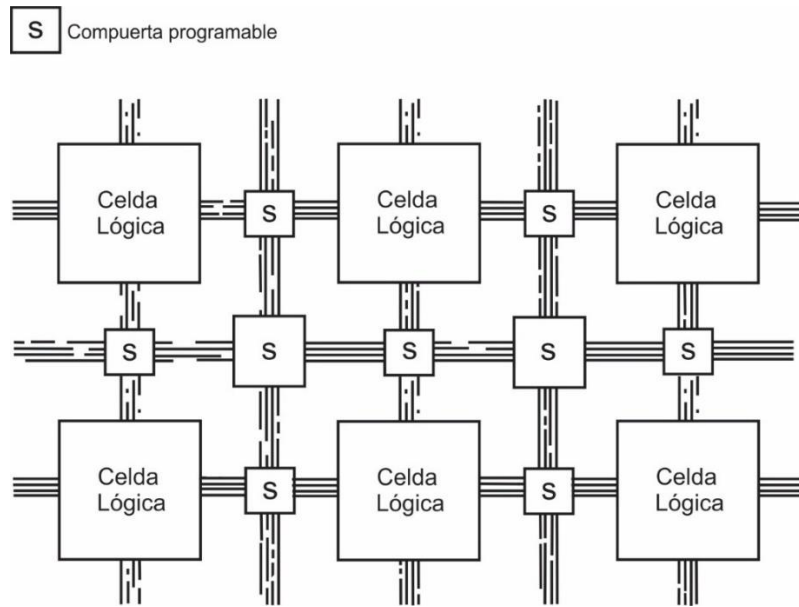


Figura N°2.6: Estructura conceptual de un dispositivo FPGA.

*Fuente: Elaboración Propia*

Un FPGA tiene tres elementos principales, Look Up Tables (LUT), flip-flops, y la matriz de enrutamiento, todos estos trabajan en conjunto para crear un dispositivo flexible.

#### ▪ LUT

Tabla de búsqueda o LUT es la forma en que la lógica de un FPGA se implementa en realidad. Un LUT consta de un número determinado de entradas y una salida, que consiste en un bloque de memoria RAM que está indexado por las entradas de la LUT. La salida de la LUT es cualquier valor que se encuentra en la ubicación indizada que está en la memoria RAM. Por ejemplo, en la figura 2.7 observamos un LUT de dos entradas.

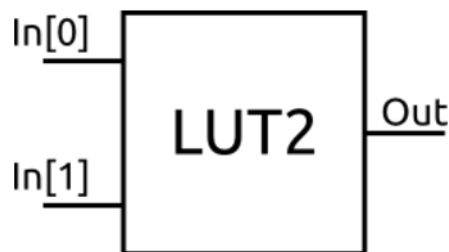


Figura N°2.7: Estructura LUT de dos entradas.

*Fuente: Elaboración Propia*

Desde la RAM en la LUT se puede ajustar a cualquier cosa, una LUT de dos entradas puede convertirse en cualquier puerta lógica.

- **Flip-Flops**

A la salida de cada LUT se puede conectar opcionalmente a un *flip-flop*. A los grupos de LUT y *flip-flops* se denominan SLICE. Estos *flip-flops* son típicamente configurables que permiten un tipo de *reset* (asíncrono y síncrono) y el nivel de *reset* (alto y bajo) lo cual tiene que ser especificado. Algunos de los flip-flops en realidad pueden ser configurados como *latches* en lugar de flip-flops, a pesar de que los *latches* no se consideran una buena práctica, ya que pueden conducir a problemas de tiempo.

- **Matriz de enrutamiento**

El siguiente bloque de tamaño en la FPGA es Complex Logic Block (CLB) y cada CLB consiste en dos SLICES. Cada CLB se conecta a una matriz de enrutamiento que es responsable de la conexión de la CLB para el resto de la FPGA. La matriz de enrutamiento puede conectar las entradas y salidas del CLB a la *matriz de enrutamiento* en general o bien entre sí. De esa manera la salida de un LUT puede introducirse en la entrada de otro LUT sin realizar una ruta larga.

### 2.2.8. VHDL

Para Maxinez (2014), El lenguaje de programación VHDL (Very Hardware Description Language) constituye una de las herramientas de programación con mayor uso en el ambiente industrial y en el ámbito universitario, debido a la versatilidad con la cual se pueden describir, sintetizar circuitos y sistemas digitales en la búsqueda de soluciones de aplicación inmediata.

Las opciones que brinda el hardware para la integración de la aplicación permiten introducir y utilizar el lenguaje en diversas materias, cuyo nivel de profundidad y diseño depende de los bloques lógicos por emplear; de esta manera, el diseño lógico, sistemas digitales, la robótica, la arquitectura de computadoras y la algorítmica, entre otras, son algunas de las áreas donde VHDL, presenta una

alternativa de diseño. Por ejemplo, en un curso de diseño lógico, quizá lo más recomendable sea utilizar el lenguaje y programar en dispositivos GAL; sin embargo, esta no es una condición necesaria e indispensable, pues depende en gran medida de la economía personal o institucional.

Este lenguaje de descripción en Hardware VHDL se estructura en módulos o unidades funcionales, identificados mediante una palabra reservada y particular de este lenguaje. En tanto, a su vez, cada módulo tiene una secuencia de instrucciones o sentencias, las cuales, en conjunto con las declaraciones de las unidades involucradas en el programa, permiten la descripción, la comprensión, la evaluación y la solución de un sistema digital. A continuación, se describe los principales módulos funcionales que se deben tener en cuenta:

- **Bibliotecas (Library)**

Las descripciones en VHDL se almacenan en archivos de diseño, los cuales contienen las sentencias que conforman los distintos módulos que integran el sistema. El compilador es quien se encarga de analizar, compilar e incluir en un archivo llamado biblioteca o con la palabra clave en VHDL *library*.

- **Entidad (Entity)**

Se define en VHDL con la palabra clave *entity*, que representa la caracterización del dispositivo físico, muestra las entradas y salidas que el diseñador ha considerado pertinentes para integrar su aplicación. Es decir, constituye un bloque de diseño que puede ser analizado y programado como un elemento individual, ya sea como un bloque lógico o como un sistema considerando entradas y salidas.

- **Arquitectura (Architecture)**

Las entidades al ser declaradas tienen asociados diferentes estilos. Estos cuerpos se denominan arquitectura o con la palabra clave en VHDL *ARCHITECTURE*.



Para el desarrollo de sistemas digitales con un lenguaje de descripción de hardware se debe tener en cuenta los siguientes niveles de abstracción, según Chu (2006) nos indica la siguiente división.

- **Nivel transistor**

Es el nivel más bajo de abstracción. A este nivel los bloques básicos de construcción son los transistores, resistencias, capacitores, etc. La descripción de su funcionamiento se basa en diferentes ecuaciones diferenciales o algún tipo de diagrama corriente-voltaje. En este nivel, un circuito digital es tratado como un sistema analógico, en el que las señales son variables en el tiempo y pueden asumir cualquier valor de un rango continuo.

La descripción física de este nivel comprende la disposición física y detalla de los componentes y sus interconexiones, siendo esto el resultado final del diseño.

- **Nivel compuerta**

Este nivel incluye en su construcción bloques lógicos simples de compuertas, como and, or, xor y multiplexores; y elementos básicos de memoria, como latches, flip-flops. En lugar de usar valores continuos utiliza valores discretos de señales de voltaje altos y bajos, interpretados mediante símbolos lógicos 1 o 0 respectivamente.

La descripción física de este nivel es la disposición de compuertas y enrutamiento de las interconexiones.

- **Abstracción RTL (Register Transfer Level)**

El nivel de registro-transferencia o más conocido por sus siglas en inglés RTL, incluye los bloques básicos de construcción modulares de compuertas simples. Incluyen unidades funcionales, como registros, y componentes de enrutamiento de datos, como multiplexores.

La principal característica de la descripción a nivel RT es el uso de una señal de reloj común en las componentes de almacenamiento. La señal de reloj funciona

como un muestreo y un pulso de sincronización, poniendo los datos en el componente en algún punto en particular que normalmente es en la caída o bajada de la señal de reloj. En un sistema correctamente diseñado, el periodo de reloj es lo suficientemente largo de manera que todas las señales de datos se estabilizan dentro del periodo de reloj. Dado que las señales de datos son muestreadas solo en el borde del reloj, la diferencia en los retardos de propagación y los fallos no tienen impacto en el funcionamiento del sistema.

La descripción física de este nivel se le conoce como *floor plan* o plano de planta. Es útil para encontrar el camino más lento entre los componentes de almacenamiento y para determinar el periodo de reloj.

- **Nivel de procesador**

Este es el nivel más alto de abstracción. Los elementos básicos para la construcción de bloques son, frecuentemente conocidos como propiedades intelectuales (IF), incluye procesadores, módulos de memoria, interfaces de buses y así sucesivamente. La descripción del comportamiento de un sistema es como el código de un programa en un lenguaje convencional, incluyendo pasos computacionales y procesos de comunicación. Las señales son agrupados e interpretadas como varios tipos de datos. La medición del tiempo es expresada en términos de pasos computacionales, que se compone en un conjunto de operaciones definido entre dos puntos de sucesivos de sincronización. Unas colecciones de ejecuciones computacionales pueden realizarse en hardware de manera paralela e intercambiar datos a través de un protocolo de comunicación o de bus ya definido.

La descripción física de un sistema a este nivel también se le conoce como *floor plan* o plano de planta, por supuesto que los componentes utilizados son mayores que los usados en un sistema de nivel RT.

### **2.2.9. Software para simulación de Lenguaje de Descripción de Hardware**

Actualmente existen muchos simuladores para lenguajes de descripción de hardware como VHDL o Verilog, los cuales pertenecen a las diferentes compañías desarrolladoras de FPGA. A continuación se mencionan los más utilizados.

- ISE Simulator (ISIM)
- GHDL
- ModelSim

#### **2.2.9.1 ISE Simulator (ISIM)**

Isim ofrece un simulador de HDL completa, con todas las funciones integradas en ISE (Xilinx, 2017). El simulador de HDL que viene integrado en el software ISE Design Suite v14.7 de la marca de FPGA XILINX, la cual se utilizará en este proyecto por ser la más representativa en el mercado.

#### **2.2.10. Sistemas Embebidos**

Este término hace referencia a todo circuito electrónico digital capaz de realizar operaciones de computación, generalmente en tiempo real, que sirven para cumplir una tarea específica en un producto. Suele tener recursos limitados y aplicaciones específicas que los hacen sumamente útiles en múltiples ambientes, como en el campo automotriz, sistemas telefónicos, robótica, instrumentación industrial, equipos médicos, entre otros.

La arquitectura de un sistema embebido contiene un microprocesador dedicado capaz de ejecutar instrucciones a una determinada velocidad, controlada por una señal de reloj. De acuerdo con la arquitectura del microprocesador del sistema embebido, los recursos internos (periféricos) y la máxima frecuencia de operación, se define la potencia de procesamiento, normalmente, este parámetro se mide en unidades de MIPS (millones de instrucciones por segundo).

## **2.3. Definición de términos**

### **2.3.1. Articulación robótica**

Es la unión entre los elementos estructurales rígidos denominados enlaces o eslabones, de la extremidad de un robot, la cual permitirá el movimiento relativo de cada dos enlaces consecutivos.

### **2.3.2. Grado de libertad**

Grado de libertad (DOF, por sus siglas en inglés), es el movimiento que puede realizar la articulación robótica, como giros y desplazamientos. Son los parámetros que se precisan para determinar la posición y la orientación del elemento final de un manipulador o extremidad. El número de grados de libertad del robot viene dado por la suma de los grados de libertad de las articulaciones que lo componen.

### **2.3.3. Robot hexápodo**

Es un vehículo mecánico capaz de caminar sobre 6 patas, el cual puede ser estable en 3 o más patas, al poseer una gran flexibilidad de adaptación si es que una de ellas fallara.

### **2.3.4. Arquitectura embebida**

Es un sistema cuya función es controlada por un computador integrado, el cual puede ser un microprocesador o microcontrolador. Implica que se encuentra dentro de un sistema general oculto a la vista y forma parte de un todo de mayores dimensiones.

### **2.3.5. Concurrencia**

Propiedad de un sistema en el cual los procesos computacionales se realizan de manera simultánea e interactuando entre ellos. Las operaciones pueden ser ejecutados en múltiples procesadores, o ejecutados en procesadores separados físicamente o virtualmente en diferentes ciclos de ejecución. Se hace cargo de establecer cómo se definen distintas tareas, que características tienen y cómo se implementan sobre un hardware en concreto. También establece los mecanismos de coordinación y sincronización necesarios para lidiar con la indeterminación, ya

que no sabemos cuándo terminarán las tareas ni cuando acceden a las estructuras de datos compartidas.

#### **2.3.6. Paralelismo**

Es un método de programación en donde varios cálculos se pueden realizar simultáneamente. Se preocupa de analizar cómo superponer operaciones con el objeto de mejorar el rendimiento al realizar una tarea concreta.

#### **2.3.7. Área de trabajo**

Es el grupo de puntos en el espacio que puede ser alcanzados por el efector final.

# **CAPITULO III**

### 3. MATERIAL Y MÉTODO

#### 3.1. Material

##### 3.1.1. Población

Arquitecturas embebidas para aplicaciones robóticas.

##### 3.1.2. Muestra

Arquitecturas embebidas para robots caminantes basadas en FPGA.

##### 3.1.3. Unidad de Análisis

Se busca hacer un análisis de la cinemática inversa y del algoritmo CORDIC en una extremidad robótica hexápoda de tres grados de libertad para el diseño de una entidad embebida de cálculo en FPGA.

#### 3.2. Método

##### 3.2.1. Nivel de Investigación

Aplicada

##### 3.2.2. Diseño de Investigación

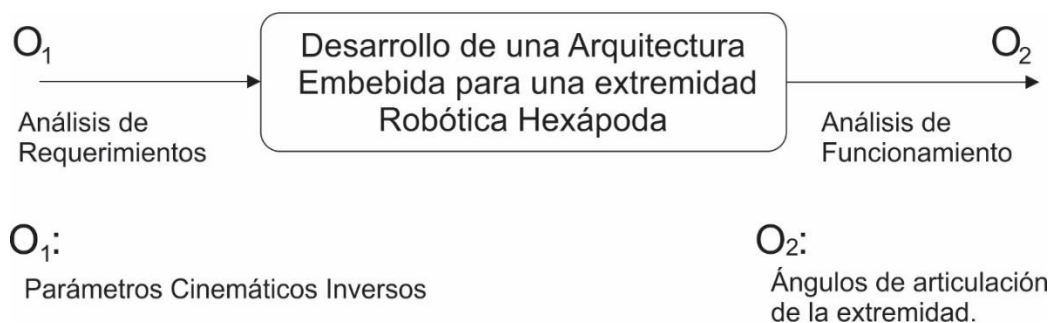


Figura N°3.1: Diseño de la Investigación.

*Fuente: Elaboración propia*

##### 3.2.3. Variables de estudio y operacionalización

###### Variables.

###### a) Variable Independiente:

Diseño de la arquitectura embebida.

Indicadores:

- Recursos utilizados.
- Esquema RTL.
- Velocidad de procesamiento.

**b) Variable Dependiente:**

Cálculo cinemático inverso.

Indicadores:

- Parámetros cinemáticos inversos.
- Precisión.

**Definición operacional.**

Tabla N°3.1. Operacionalización de la Variable Independiente

VARIABLE INDEPENDIENTE	DEFINICION CONCEPTUAL	DEFINICION OPERACIONAL	INDICADORES	INSTRUMENTOS	UNIDADES MEDIDA
Diseño de la arquitectura embebida en un FPGA	Es el análisis y modelamiento de un sistema digital descrito en lenguaje VHDL.	Es la cuantificación del sistema digital modelo en lenguaje VHDL mediante el uso de los recursos de hardware y la velocidad de procesamiento para lograr los resultados esperados.	Recursos utilizados	Reporte de simulación	LUT
			Estructura RTL		-----
			Velocidad de procesamiento		μs

*Fuente: Elaboración Propia*



Tabla N°3.2. Operacionalización de la Variable Dependiente

VARIABLE DEPENDIENTE	DEFINICION CONCEPTUAL	DEFINICION OPERACIONAL	INDICADORES	INSTRUMENTOS	UNIDADES MEDIDA
Cálculo cinemático inverso	Es la cuantificación de datos específicos de los parámetros dados por las ecuaciones de la cinemática inversa.	La cuantificación de los parámetros y la precisión del cálculo cinemático inverso mediante la simulación de diferentes ángulos de trabajo obteniendo resultados computacionales realizados en software y hardware.	Parámetros cinemáticos inversos	Reporte de simulación	Radianes (rad)
			Precisión	Cuadro comparativo	-----

*Fuente: Elaboración Propia*

### 3.2.4. Técnicas e instrumentos de recolección de datos

#### a) Actuador en aplicaciones robóticas

Teniendo en cuenta que, la precisión de los ángulos cinemáticos obtenidos mediante la arquitectura propuesta en FPGA con respecto al cálculo en Matlab sea apreciable se debe considerar las características técnicas del actuador, pero principalmente la resolución, esto se muestra en la tabla 3.3 en donde se ha realizado una comparación entre diferentes marcas de actuadores del mercado, señalando sus principales características como el rango de voltaje de operación, torque, velocidad, resolución tipo y ángulo de operación.

El actuador dependerá de la aplicación robótica elegida y este influirá directamente en el error porcentual de la arquitectura propuesta.

Tabla N°3.3: Cuadro comparativo de actuadores.

Marca	Tower Pro	Hitec	Dynamixel	Herkulex
Modelo	MG996R	HS-645MG	AX-12	DRS-0201
Voltaje Operación	4.8 – 7.2VDC	4.8 – 6VDC	9-12VDC	7-12VDC
Torque	11kf.cm	7.7kf/cm	15.3kg.cmm	24kgf.cm
Velocidad sin carga	0.14s/60°	0.24s/60°	0.169s/60°	0.147s/60°
Resolución	1°	1°	0.29°	0.325°
Tipo	Analógico	Analógico	Digital	Digital
Ángulo de operación	120°	360°	300°	320°

*Fuente: Elaboración propia.*

#### **b) Ecuaciones de cinemática inversa**

Considerando el trabajo realizado por Evangelista (2014) en sus estudios de una plataforma de investigación robótica, se considera el siguiente sistema de referencia de una extremidad robótica hexápoda de tres grados de libertad, la cual es la base para obtener las ecuaciones cinemáticas (figura 3.2).

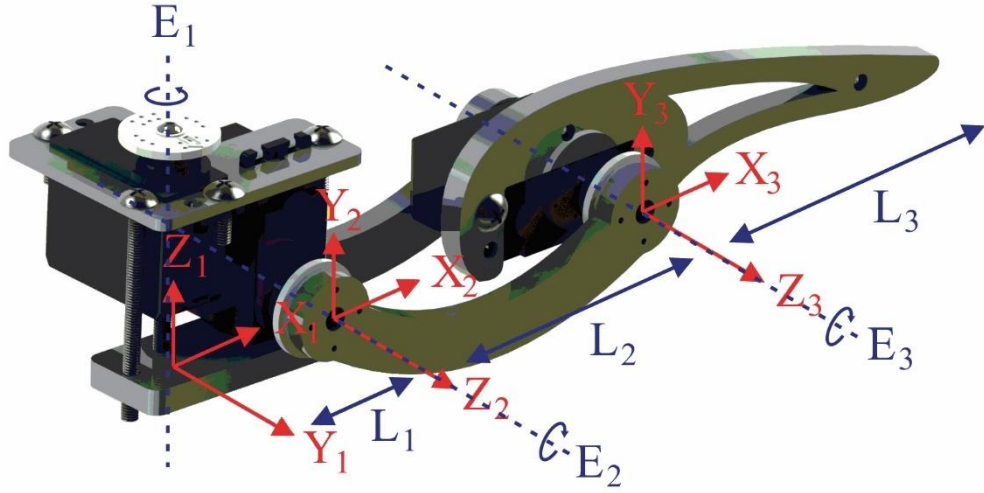


Figura N°3.2: Sistema de referencia de una extremidad robótica.

*Fuente: G. Evangelista (2014) "Design and Modeling of a Mobile Research Platform based on Hexapod Robot with Embedded System and Interactive Control".*

En la figura anterior, podemos observar que se considera  $L_1$ ,  $L_2$  y  $L_3$  como las medidas de cada enlace. Además de los ángulos de articulación para los tres grados de libertad,  $\theta_1$ ,  $\theta_2$  y  $\theta_3$ , representados por los ejes  $E_1$ ,  $E_2$  y  $E_3$  respectivamente.

Del sistema de referencia, se obtienen las ecuaciones 3.1, 3.2 y 3.3, que representan las ecuaciones de la cinemática inversa de la extremidad robótica. Las cuales, han sido reformuladas con base a las ecuaciones 2.1, 2.2 y 2.3, con el fin de adaptarse a los operadores de CORDIC. Donde las variables  $x_e$ ,  $y_e$ ,  $z_e$  representan los valores de la coordenada de posición del elector final de la extremidad robótica hexápoda en la cual la extremidad se va a ubicar según los valores de los ángulos de articulación.

$$\theta_1 = \text{atan}(y_e/x_e) \quad (3.1)$$

$$\theta_2 = \text{atan}\left(\frac{G}{\sqrt{1-G^2}}\right) - \text{atan}\left(\frac{\sin \theta_3}{F + \cos \theta_3}\right) \quad (3.2)$$

$$\theta_3 = \text{atan}\left(\frac{\sqrt{1-D^2}}{D}\right) \quad (3.3)$$

Estas ecuaciones se dividen en variables para un desarrollo por segmentos de la arquitectura. Las cuales se muestran en las ecuaciones 3.4 al 3.11.

$$r = \sqrt{x_e^2 + y_e^2} \quad (3.4)$$

$$A = 2l_1 r \quad (3.5)$$

$$B = \sqrt{(r - l_1)^2 + z_e^2} \quad (3.6)$$

$$C = r^2 + z_e^2 + l_1^2 - l_2^2 - l_3^2 - A \quad (3.7)$$

$$D = C \quad C_a = \cos \theta_3 \quad (3.8)$$

$$G = \frac{z_e}{B} \quad (3.9)$$

Parámetros:

$$C_a = \frac{1}{2l_2 l_3} \quad (3.10)$$

$$F = \frac{l_2}{l_3} \quad (3.11)$$

Siendo los valores de los enlaces de la extremidad, tomando como ejemplo el prototipo robótico hexápodo de G. Evangelista (2014). Los cuales se utilizan para la simulación de la arquitectura en el software ISim.

$$L_1 = 0.0275 \text{ m} \quad (3.12)$$

$$L_2 = 0.0963 \text{ m} \quad (3.13)$$

$$L_3 = 0.1051 \text{ m} \quad (3.14)$$

En la presente investigación se considera metros ( $m$ ) como unidad de medida del desarrollo cinemático, porque el rango de convergencia de las operaciones realizadas en CORDIC es limitado, lo cual será ampliado en el apartado b.

### c) Delimitaciones para el diseño de la arquitectura CORDIC

#### Área de trabajo

En el presente desarrollo se analizará el área de trabajo generada por los tipos de caminata trípode, cuadrúpeda y hexápoda presentada en Guillermo (2014).

La trayectoria del efector final de la extremidad robótica limita el movimiento de las articulaciones en un área representada por la figura 3.3. La cual limitará los valores de rotación máximos requeridos para el diseño del operador CORDIC.

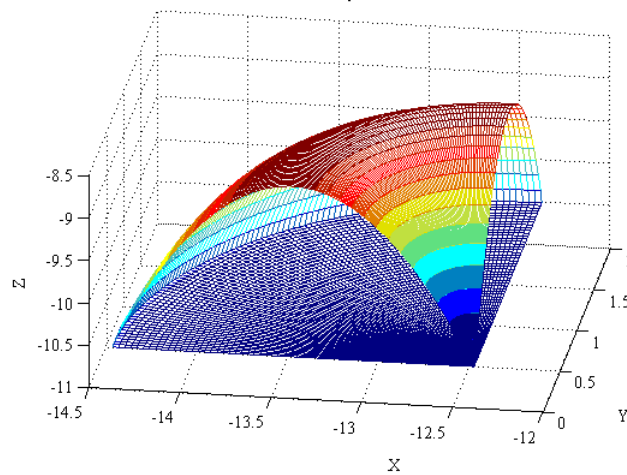


Figura N°3.3: Área del rango de trabajo del efector final.

*Fuente: G. Evangelista (2013) “Desarrollo y construcción de una plataforma de investigación robótica hexápodo mediante el uso de un dispositivo MBED NXP LPC1768”.*

### **Delimitación del rango de convergencia**

Hu (1991) describe un análisis de cada método CORDIC presentado en la tabla 3.4 incluyendo el rango básico de convergencia obtenido a partir de las formulaciones de la tabla 2.1.

Tabla N°3.4. Análisis Rango de Convergencia CORDIC

Método	Rango de convergencia	
Circular	$ z_{in}  \leq \tan^{-1}(2^{-N}) + \sum_{i=0}^N \tan^{-1}(2^{-i})$	$ z_{in}  \leq 1.7433(99.9^\circ) = \theta_{max}$
Lineal	$ z_{in}  \leq \sum_{i=0}^N (2^{-i})$	$ z_{in}  \leq \theta_{max} \approx 2 (114.59^\circ)$
Hiperbólico	$ z_{in}  \leq \tanh^{-1}(2^{-N}) + \sum_{i=1}^N \tanh^{-1}(2^{-i})$	$ z_{in}  \leq \theta_{max} \approx 1.1182 (64.07^\circ)$

*Fuente: Elaboración propia*

Así, los valores de  $z_{in}$  para cada modo del operador CORDIC que satisfacen el área de trabajo de la extremidad robótica y la tabla 3.4 indican si el operador requiere una expansión para cada uno de sus modos.

De la tabla 2.1, se aprecia que el operador CORDIC hiperbólico vectorial es utilizado para desarrollar la ecuación 3.3. Siendo esto el punto de partida para el análisis de la delimitación del rango de convergencia de este operador. Así como, observamos que el rango de convergencia no sobrepasa los valores de 1.7433 y no es menor que 1.1182 en el caso del hiperbólico, por tal motivo se eligió los valores de los enlaces 11, 12 y 13 en metros para que se mantengan en rango de trabajo.

#### **d) Expansión del Rango de Convergencia CORDIC**

Según las ecuaciones 3.1, 3.2 y 3.3, en la presente investigación solo se utiliza los sistemas de rotación circular e hiperbólico. El modo lineal no se utiliza debido a que las operaciones de multiplicación y división son realizadas por entidades de multiplicación por hardware para punto flotante.

Para solucionar el problema del rango, se ha seleccionado el primer método planteado por Hu (1991), ya que implica el uso de menos recursos lógicos. El método indica que se deben agregar iteraciones negativas, donde las cantidades de estas se tienen en cuenta según la tabla 3.5.

Tabla N°3.5.  $\theta_{\text{máx}}$  obtenido del nuevo rango de convergencia

-i	$\theta_{\text{máx}}$
0	2.09113
1	3.44515
2	5.16215
3	7.23371
4	9.6581
5	12.42644
6	15.54462
7	19.00987
8	22.82194
9	26.9807
10	31.48609

*Fuente: Elaboración Propia.*

Este método consiste en reemplazar la tangente hiperbólica por la expresión de la ecuación 3.15 para las iteraciones adicionales, donde  $i \leq 0$ .

$$\tanh(\alpha_i) = 1 - 2^{i-2} \quad (3.15)$$

Al tener mayores iteraciones adicionales el rango de convergencia también aumenta, como se observa en la tabla 3.5. En 3.16 se muestra como hallar el  $\theta_{\text{máx}}$ , donde -M es el máximo número de iteraciones adicionales y N es el máximo número de iteraciones normales.

$$\theta_{m\acute{a}x} = \sum_{i=-M}^0 \tanh^{-1}(1-2^{i-2}) + \tanh^{-1}(2^{-n}) + \sum_{i=1}^N \tanh^{-1}(2^{-i}) \quad (3.16)$$

Finalmente, la ecuación para cada rotación para la parte de expansión se define en el sistema de ecuaciones 3.17 la cual se utilizará para el desarrollo del algoritmo en el simulador.

$$\begin{aligned} x_{i+1} &= x_i - d_i y_i (1-2^{i-2}) \\ y_{i+1} &= y_i + d_i x_i (1-2^{i-2}) \\ z_{i+1} &= z_i - d_i \tanh^{-1}(1-2^{i-2}) \end{aligned} \quad (3.17)$$

En este trabajo se utiliza 22 iteraciones con un rango de -5:16 donde M = -5 y N = 16, según el análisis desarrollado por Agurto (2006). Considerando el número de iteraciones se obtiene el valor de  $K_h = 0.00050283$ , obtenido según la ecuación 3.18 planteada por Hu (1991).  $5.0283 \cdot 10^{-4}$  i  $K_h = 1988.7437$

$$K_h = \left[ \prod_{i=-M}^0 \sqrt{1-(1-2^{-2^{i+1}})^2} \right] \left[ \prod_{i=1}^N \sqrt{1-(2^{-i})^2} \right] \quad (3.18)$$

## e) Diseño de la entidad CORDIC

### Operadores de CORDIC

En este trabajo, se considera tres tipos de operadores de CORDIC como son circular rotacional, circular vectorial e hiperbólico vectorial, cuyas funciones y simbología se representa en la tabla 3.6.

Tabla N°3.6. Funciones y símbolos de los operadores CORDIC

Circular Rotacional (CR)	Circular Vectorial (CV)	Hiperbólico Vectorial (HV)
$\begin{array}{l} x \rightarrow \boxed{\text{CR}} \rightarrow K(x \cos z - y \sin z) \\ y \rightarrow \boxed{\text{CR}} \rightarrow K(y \cos z + x \sin z) \\ z \rightarrow \boxed{\text{CR}} \rightarrow 0 \end{array}$	$\begin{array}{l} x \rightarrow \boxed{\text{CV}} \rightarrow K \sqrt{x^2 + y^2} \\ y \rightarrow \boxed{\text{CV}} \rightarrow 0 \\ z \rightarrow \boxed{\text{CV}} \rightarrow z + \tan^{-1}(y/x) \end{array}$	$\begin{array}{l} x \rightarrow \boxed{\text{HV}} \rightarrow K' \sqrt{x^2 - y^2} \\ y \rightarrow \boxed{\text{HV}} \rightarrow 0 \\ z \rightarrow \boxed{\text{HV}} \rightarrow z + \tanh^{-1}(y/x) \end{array}$

*Fuente: Elaboración Propia*

#### ▪ Circular Rotacional (CR)

Para obtener el valor de tangente considerando la siguiente identidad.



$$\tan(z) = \sin z / \cos z$$

▪ **Circular Vectorial (CV)**

Para obtener el valor arco-tangente y la raíz cuadra de la suma de dos variables al cuadrado.

▪ **Hiperbólico Vectorial (HV)**

Se utiliza este algoritmo para poder hallar la raíz cuadra de la diferencia de los cuadros de dos números, los cuales serán x e y. Para lo cual se debe tener en cuenta que el rango de convergencia se ha expandido según lo señalado anteriormente.

### Arquitectura del Algoritmo CORDIC

En la presente investigación se realiza una comparación entre las diferentes arquitecturas propuestas implementable en hardware según Schweers (2002). Señalando las principales arquitecturas como, bit-paralela iterativa, bit-paralela desplegada y bit serie-iterativa, las cuales son descritas en la tabla 3.7.

Tabla N°3.7. Tipos de Arquitecturas para implementar CORDIC

ARQUITECTURA	CONCLUSIÓN
Bit-Paralela Iterativa	Uso eficiente del hardware, porque los recursos son reutilizados en cada iteración.
Bit-Paralela Desplegada	Enorme cantidad de espacio para su implementación.
Bit-Serie Iterativa	Mayor tiempo de procesamiento, así como el requerimiento de hardware de control más complejo.

*Fuente: Elaboración Propia*

En base a esto, se opta por utilizar la arquitectura bit paralela-iterativa debido a las ventajas presentadas visualizadas en la tabla anterior, las cuales son apropiadas para optimizar el uso de recursos de hardware en FPGA.

### Formato Punto Flotante

El formato utilizado en la presente investigación para los tipos de datos numéricos y operaciones es el IEEE-754 (2008). De la figura 2.3, se aprecia que el esquema de arquitectura bit-paralela iterativa se basa en dos operaciones adición y desplazamiento binario. Para resolver la adición se diseña una entidad de adición en punto flotante y para los desplazamientos binarios se resta una cantidad definida al exponente del número, según el Departamento de Ciencias Computacionales de la Universidad de Wisconsin-Madison (2015).

### Entidad CORDIC

En base a lo descrito anteriormente, se diseña la entidad CORDIC con las señales de entrada y salida descritas en la figura 3.4.

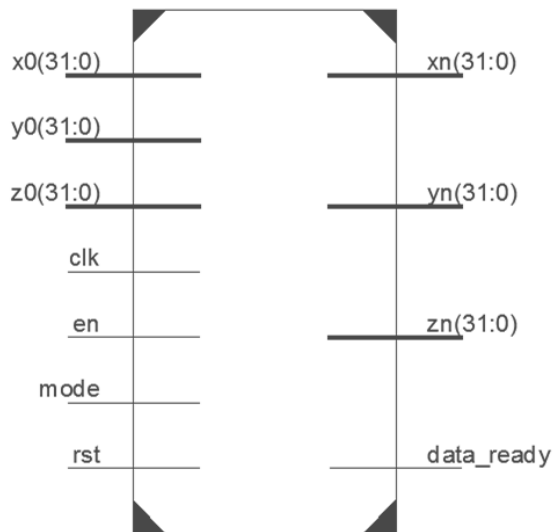


Figura N°3.4. Diseño de entidad CORDIC.

*Fuente: Elaboración Propia*

ENTRADAS	{	$x_0, y_0, z_0$ (Vector 32 bits): Parámetros de entrada.
		Clk (Booleano): Señal de reloj.
		En (Booleano): Habilitador.
		Mode (Booleano): Selección de modo de operación.
		Rst (Booleano): Reset.

SALIDAS	$x_n, y_n, z_n$ (Vector 32 bits): Parámetros de salida.
	Data_ready (Booleano): Indica cuando los parámetros de salida son datos válidos.

#### f) Cuadro comparativo de recursos de un FPGA

Para la elección del dispositivo en el cual se realizará la simulación, se ha elegido a la familia del Spartan-3E FPGA, el cual se detalla sus características según cada dispositivo de esta familia en la tabla 3.8.

Tabla N°3.8. Tabla de requerimiento de hardware del FPGA Spartan-3E

DISPOSITIVO	CLB	SLICE	LUT	FLIP- FLOPS
XC3S100E	240	960	1920	1920
XC3S250E	612	2448	4896	4896
XC3S500E	1164	4656	9312	9312
XC3S1200E	2168	8672	17344	17344
XC3S1600E	3688	14752	29504	29504

*Fuente: Elaboración Propia*

Se elige el modelo XC3S500E, ya que es el modelo con el cual se cuenta en los laboratorios de la escuela de Ingeniería Electrónica de la Universidad Privada Antenor Orrego. No obstante, la simulación en el ISE Design Suite en lenguaje VHDL facilita la migración a cualquier familia o dispositivo FPGA según las necesidades.

#### g) Diseño de la arquitectura para el cálculo cinemático inverso a nivel RTL

De estos operadores se realiza el diagrama de bloques de la arquitectura propuesta para poder calcular las ecuaciones de la cinemática inversa 3.1, 3.2 y 3.3. La cual se observa en la figura 3.5 mediante bloques de funciones (FBs) basados en los operadores de CORDIC de la tabla 3.5.

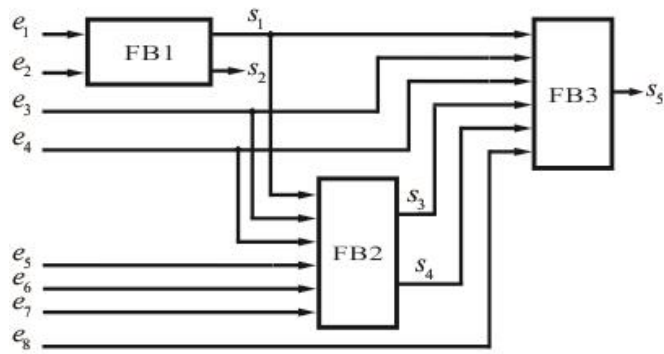


Figura N°3.5: Diagrama de bloques para la realización de la cinemática inversa en FPGA.

*Fuente: Elaboración propia*

La estructura CORDIC en FPGA de FB1, esta basa en la ecuación 3.1 para hallar el primer ángulo de articulación  $\theta_1$ . En la figura 3.6 se describe la estructura interna de FB1, en el cual  $e_1 = x_e$ ,  $e_2 = y_e$  son las entradas obteniendo  $S_1 = r$  y  $S_2 = \theta_1$ , donde  $iK$  representa a la inversa del factor de expansión ( $iK = 1.6467$ ).

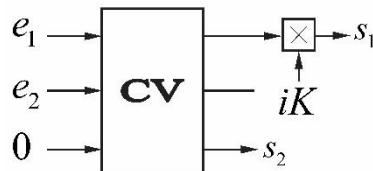


Figura N°3.6: Diagrama interno del bloque FB1.

*Fuente: Elaboración propia*

Según las ecuaciones 3.2 y 3.3, primero se debe desarrollar las ecuaciones del 3.3 para hallar primero el ángulo de la tercera articulación  $\theta_3$  y luego 3.2 para hallar  $\theta_2$ , ya que esta última depende del resultado del tercer ángulo de articulación. El diagrama interno de FB2 representada por la figura 3.7 basada en la ecuación 3.3 se obtiene como resultado  $S_4 = \theta_3$ , que tiene como entradas  $S_1 = r$ ,  $e_3 = Z_e$ ,  $e_4 = l_1$ ,  $e_5 = l_2$ ,  $e_6 = l_3$ ,  $e_7 = Ca$ . Donde  $iKh$  es la inversa del factor de expansión para el CORDIC Hiperbólico (HV),  $iKh = 47,6216$ .

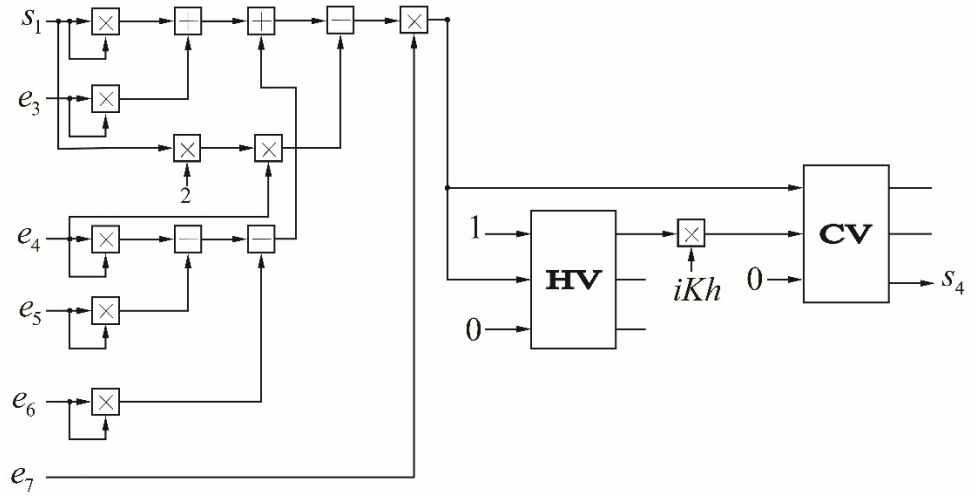


Figura N°3.7: Diagrama interno del bloque FB2.

*Fuente: Elaboración propia*

Luego de obtener el valor de  $\theta_3$ , se diseña el bloque FB3 descrito en la figura 3.8 donde se obtiene  $S_5 = \theta_2$ , cuyas entradas son  $S_1 = r$ ,  $e_4 = 11$ ,  $e_3 = ze$ ,  $S_3 = D$ ,  $e_8 = F$ ,  $S_4 = \theta_3$ .

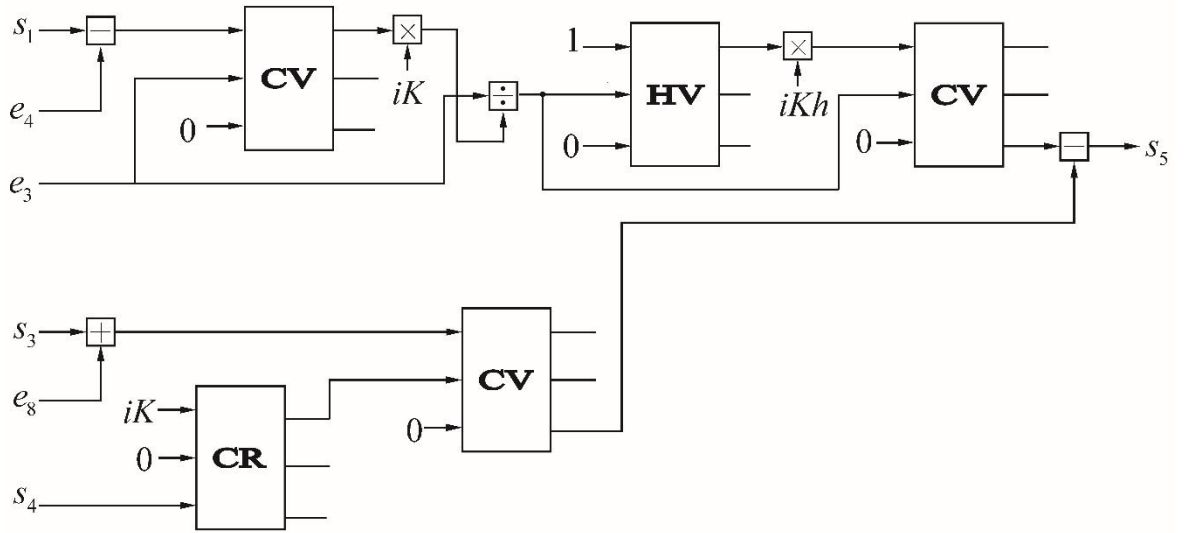


Figura N°3.8: Diagrama interno del bloque FB3.

*Fuente: Elaboración propia*

La propuesta de arquitectura de la figura 3.9 presenta los bloques CC1, CC3, CC4, CC5, CC6, CC9 los cuales aplica el CORDIC circular en modo rotacional o vectorial y el CH2, CH8 que son bloques de CORDIC hiperbólico. Además, se

diferencia la máquina de estado que se utiliza para realizar los desplazamientos y operaciones aritméticas necesarias para hallar  $D$  y  $r$ .

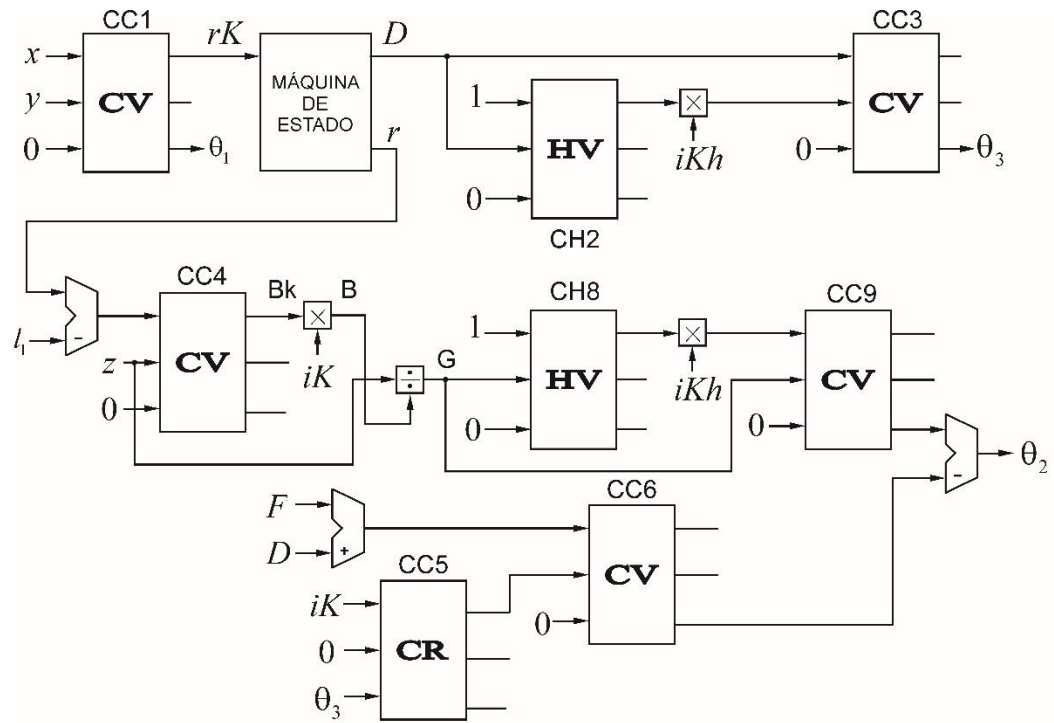


Figura N°3.9: Arquitectura propuesta para cinemática inversa.

*Fuente: Elaboración propia*

#### h) Reporte de simulación obtenido del software Matlab R2015b-MathWorks.

En base a una muestra de 30 iteraciones, aplicando el algoritmo en Matlab del anexo 01, donde las posiciones iniciales ( $x_0$ ,  $y_0$  y  $z_0$ ) y los parámetros considerados para la trayectoria de cada tipo de caminata hexápoda son datos obtenidos de los resultados de Guillermo (2013), se han construido las tablas 3.8 a 3.11 donde se muestran los ángulos de articulación de la cinemática inversa.

El tiempo de ejecución para la obtención de  $\theta_1$ ,  $\theta_2$ , y  $\theta_3$ , es de 0.003193 segundos.

## Caminata Trípode

Tabla N°3.9. Ángulos calculados por Matlab/Caminata Trípode

$i$	Posición Calculada (m)			Ángulos Calculados (rad)		
	$x_1$	$y_1$	$z_1$	$\theta_1$	$\theta_2$	$\theta_3$
0	0.124	0.050	-0.105	0.3838	-0.0047	-1.4735
1	0.124	0.052	-0.099	0.3954	0.0592	-1.5297
2	0.124	0.053	-0.094	0.4068	0.1135	-1.5736
3	0.124	0.055	-0.089	0.4181	0.1586	-1.6066
4	0.124	0.057	-0.086	0.4293	0.1943	-1.6299
5	0.124	0.058	-0.083	0.4403	0.2208	-1.6446
6	0.124	0.060	-0.081	0.4513	0.2379	-1.6512
7	0.124	0.062	-0.080	0.4621	0.2459	-1.6503
8	0.124	0.063	-0.080	0.4729	0.2445	-1.6420
9	0.124	0.065	-0.081	0.4835	0.2340	-1.6264
10	0.124	0.067	-0.083	0.4940	0.2144	-1.6033
11	0.124	0.068	-0.086	0.5043	0.1857	-1.5721
12	0.124	0.070	-0.089	0.5146	0.1478	-1.5323
13	0.124	0.072	-0.094	0.5248	0.1008	-1.4827
14	0.124	0.073	-0.099	0.5348	0.0443	-1.4219
15	0.124	0.075	-0.105	0.5447	-0.0219	-1.3480
16	0.124	0.073	-0.105	0.5348	-0.0202	-1.3581
17	0.124	0.072	-0.105	0.5248	-0.0185	-1.3680
18	0.124	0.070	-0.105	0.5146	-0.0169	-1.3776
19	0.124	0.068	-0.105	0.5043	-0.0154	-1.3869
20	0.124	0.067	-0.105	0.4940	-0.0141	-1.3960
21	0.124	0.065	-0.105	0.4835	-0.0128	-1.4049
22	0.124	0.063	-0.105	0.4729	-0.0116	-1.4135
23	0.124	0.062	-0.105	0.4621	-0.0105	-1.4218
24	0.124	0.060	-0.105	0.4513	-0.0094	-1.4299
25	0.124	0.058	-0.105	0.4403	-0.0085	-1.4378
26	0.124	0.057	-0.105	0.4293	-0.0076	-1.4454
27	0.124	0.055	-0.105	0.4181	-0.0067	-1.4528
28	0.124	0.053	-0.105	0.4068	-0.0060	-1.4599
29	0.124	0.052	-0.105	0.3954	-0.0053	-1.4669

*Fuente: Elaboración propia.*

## Caminata Cuadrúpeda

Tabla N°3.10. Ángulos calculados por Matlab/Caminata Cuadrúpeda

$i$	Posición Calculada (m)			Ángulos Calculados (rad)		
	$x_1$	$y_1$	$z_1$	$\theta_1$	$\theta_2$	$\theta_3$
0	0.1238	0.0300	-0.1051	0.2377	-0.0006	-1.5361
1	0.1203	0.0300	-0.0925	0.2444	0.1312	-1.6923
2	0.1168	0.0300	-0.0827	0.2514	0.2347	-1.8109
3	0.1133	0.0300	-0.0757	0.2588	0.3100	-1.8999
4	0.1098	0.0300	-0.0715	0.2667	0.3556	-1.9640
5	0.1063	0.0300	-0.0701	0.2751	0.3695	-2.0055
6	0.1028	0.0300	-0.0715	0.2839	0.3502	-2.0246
7	0.0993	0.0300	-0.0757	0.2934	0.2972	-2.0193
8	0.0958	0.0300	-0.0827	0.3035	0.2116	0.2116
9	0.0923	0.0300	-0.0925	0.3143	0.0963	-1.9200
10	0.0888	0.0300	-0.1051	0.3258	-0.0453	-1.8146
11	0.0906	0.0300	-0.1051	0.3199	-0.0404	-1.8033
12	0.0923	0.0300	-0.1051	0.3143	-0.0358	-1.7917
13	0.0941	0.0300	-0.1051	0.3088	-0.0314	-1.7799
14	0.0958	0.0300	-0.1051	0.3035	-0.0273	-1.7678
15	0.0976	0.0300	-0.1051	0.2984	-0.0236	-1.7554
16	0.0993	0.0300	-0.1051	0.2934	-0.0200	-1.7427
17	0.1011	0.0300	-0.1051	0.2885	-0.0167	-1.7294
18	0.1028	0.0300	-0.1051	0.2839	-0.0139	-1.7165
19	0.1046	0.0300	-0.1051	0.2793	-0.0111	-1.7027
20	0.1063	0.0300	-0.1051	0.2751	-0.0088	-1.6893
21	0.1081	0.0300	-0.1051	0.2707	-0.0067	-1.6748
22	0.1098	0.0300	-0.1051	0.2667	-0.0049	-1.6609
23	0.1116	0.0300	-0.1051	0.2626	-0.0034	-1.6459
24	0.1133	0.0300	-0.1051	0.2588	-0.0022	-1.6314
25	0.1151	0.0300	-0.1051	0.2550	-0.0012	-1.6158
26	0.1168	0.0300	-0.1051	0.2514	-0.0005	-1.6008
27	0.1186	0.0300	-0.1051	0.2478	-0.0001	-1.5846
28	0.1203	0.0300	-0.1051	0.2444	0.0000	-1.5690
29	0.1221	0.0300	-0.1051	0.2409	-0.0002	-1.5522

*Fuente: Elaboración propia.*



### Caminata Cuadrúpeda 4+2

Tabla N°3.11. Ángulos calculados por Matlab/Caminata Cuadrúpeda 4+2

$i$	Posición Calculada (m)			Ángulos Calculados (rad)		
	$x_1$	$y_1$	$z_1$	$\theta_1$	$\theta_2$	$\theta_3$
0	0.1238	0.0300	-0.1051	0.2377	-0.0006	-1.5361
1	0.1203	0.0300	-0.0925	0.2444	0.1312	-1.6923
2	0.1168	0.0300	-0.0827	0.2514	0.2347	-1.8109
3	0.1133	0.0300	-0.0757	0.2588	0.3100	-1.8999
4	0.1098	0.0300	-0.0715	0.2667	0.3556	-1.9640
5	0.1063	0.0300	-0.0701	0.2751	0.3695	-2.0055
6	0.1028	0.0300	-0.0715	0.2839	0.3502	-2.0246
7	0.0993	0.0300	-0.0757	0.2934	0.2972	-2.0193
8	0.0958	0.0300	-0.0827	0.3035	0.2116	0.2116
9	0.0923	0.0300	-0.0925	0.3143	0.0963	-1.9200
10	0.0888	0.0300	-0.1051	0.3258	-0.0453	-1.8146
11	0.0906	0.0300	-0.1051	0.3199	-0.0404	-1.8033
12	0.0923	0.0300	-0.1051	0.3143	-0.0358	-1.7917
13	0.0941	0.0300	-0.1051	0.3088	-0.0314	-1.7799
14	0.0958	0.0300	-0.1051	0.3035	-0.0273	-1.7678
15	0.0976	0.0300	-0.1051	0.2984	-0.0236	-1.7554
16	0.0993	0.0300	-0.1051	0.2934	-0.0200	-1.7427
17	0.1011	0.0300	-0.1051	0.2885	-0.0167	-1.7294
18	0.1028	0.0300	-0.1051	0.2839	-0.0139	-1.7165
19	0.1046	0.0300	-0.1051	0.2793	-0.0111	-1.7027
20	0.1063	0.0300	-0.1051	0.2751	-0.0088	-1.6893
21	0.1081	0.0300	-0.1051	0.2707	-0.0067	-1.6748
22	0.1098	0.0300	-0.1051	0.2667	-0.0049	-1.6609
23	0.1116	0.0300	-0.1051	0.2626	-0.0034	-1.6459
24	0.1133	0.0300	-0.1051	0.2588	-0.0022	-1.6314
25	0.1151	0.0300	-0.1051	0.2550	-0.0012	-1.6158
26	0.1168	0.0300	-0.1051	0.2514	-0.0005	-1.6008
27	0.1186	0.0300	-0.1051	0.2478	-0.0001	-1.5846
28	0.1203	0.0300	-0.1051	0.2444	0.0000	-1.5690
29	0.1221	0.0300	-0.1051	0.2409	-0.0002	-1.5522

*Fuente: Elaboración propia.*

## Caminata Pentápoda

Tabla N°3.12. Ángulos calculados por Matlab/Caminata Pentápoda

$i$	Posición Calculada (m)			Ángulos Calculados (rad)		
	$x_1$	$y_1$	$z_1$	$\theta_1$	$\theta_2$	$\theta_3$
0	0.1238	0.0400	-0.1051	0.3125	-0.0020	-1.5088
1	0.1244	0.0392	-0.1051	0.3051	-0.0022	-1.5056
2	0.1249	0.0384	-0.1051	0.2980	-0.0023	-1.5032
3	0.1255	0.0375	-0.1051	0.2907	-0.0026	-1.4997
4	0.1261	0.0367	-0.1051	0.2834	-0.0028	-1.4962
5	0.1267	0.0359	-0.1051	0.2761	-0.0031	-1.4926
6	0.1272	0.0351	-0.1051	0.2691	-0.0033	-1.4899
7	0.1278	0.0343	-0.1051	0.2620	-0.0036	-1.4861
8	0.1284	0.0334	-0.1051	0.2621	-0.0041	-1.4796
9	0.1290	0.0326	-0.1051	0.2477	-0.0042	-1.4783
10	0.1295	0.0318	-0.1051	0.2409	-0.0045	-1.4753
11	0.1267	0.0359	-0.0891	0.2761	0.1627	-1.6462
12	0.1238	0.0400	-0.0811	0.3125	0.2475	-1.7303
13	0.1209	0.0441	-0.0811	0.3497	0.2487	-1.7447
14	0.1181	0.0482	-0.0891	0.3874	0.1656	-1.6882
15	0.1152	0.0523	-0.1051	0.4261	-0.0004	-1.5446
16	0.1158	0.0515	-0.1051	0.4182	-0.0004	-1.5426
17	0.1163	0.0506	-0.1051	0.4107	-0.0005	-1.5413
18	0.1169	0.0498	-0.1051	0.4029	-0.0005	-1.5391
19	0.1175	0.0490	-0.1051	0.3952	-0.0006	-1.5368
20	0.1181	0.0482	-0.1051	0.3874	-0.0007	-1.5344
21	0.1186	0.0474	-0.1051	0.3800	-0.0008	-1.5328
22	0.1192	0.0466	-0.1051	0.3723	-0.0009	-1.5303
23	0.1198	0.0457	-0.1051	0.3647	-0.0010	-1.5276
24	0.1204	0.0449	-0.1051	0.3571	-0.0011	-1.5249
25	0.1209	0.0441	-0.1051	0.3497	-0.0012	-1.5231
26	0.1215	0.0433	-0.1051	0.3422	-0.0013	-1.5202
27	0.1221	0.0425	-0.1051	0.3346	-0.0015	-1.5172
28	0.1227	0.0416	-0.1051	0.3272	-0.0016	-1.5142
29	0.1232	0.0408	-0.1051	0.3199	-0.0018	-1.5120

*Fuente: Elaboración propia.*

**i) Reporte de simulación del ISE Simulator (ISim) del software ISE Design Suite v14.7 – XILINX.**

En base a una muestra de 30 iteraciones, aplicando el algoritmo en Xilinx del anexo 02 al 06, donde las posiciones iniciales (x0, y0 y z0) y los parámetros considerados para la trayectoria de cada tipo de caminata hexápoda son datos obtenidos de los resultados de Guillermo (2013), se han construido las tablas 3.12 a 3.15 donde se muestran los ángulos de articulación de la cinemática inversa.

Considerando un cristal de 50MHZ. Ciclos de reloj es entre 20ns, obteniendo los datos a continuación en base a la figura 3.10.

$\theta_1$ : 360ns

Ciclo de reloj: 18

$\theta_2$ : 2.02us

Ciclo de reloj: 101

$\theta_3$ : 1.3us

Ciclo de reloj: 65

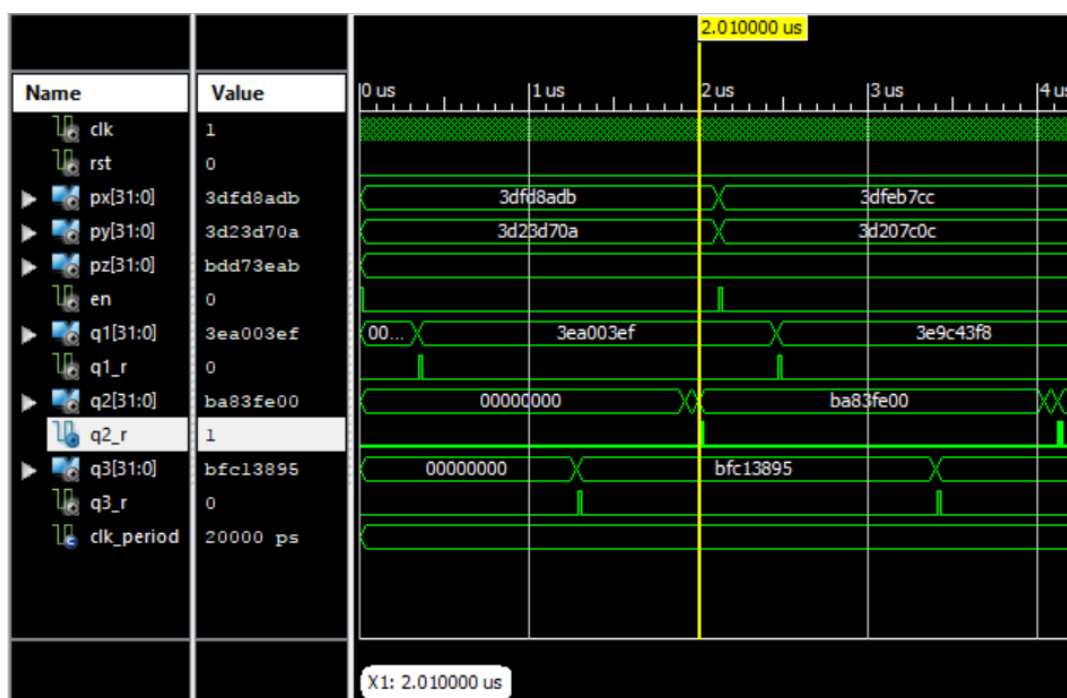


Figura N°3.10: Tiempo de ejecución de los ángulos de articulación.

Fuente: Elaboración propia

## Caminata Trípode

Tabla N°3.13. Ángulos calculados por ISim/Caminata Trípode

$i$	Posición Calculada (m)			Ángulos Calculados (rad)		
	$x_1$	$y_1$	$z_1$	$\theta_1$	$\theta_2$	$\theta_3$
0	0.1238	0.0400	-0.1051	0.3838	-0.0037	-1.4744
1	0.1244	0.0392	-0.1051	0.3954	0.0600	-1.5304
2	0.1249	0.0384	-0.1051	0.4068	0.1145	-1.5746
3	0.1255	0.0375	-0.1051	0.4181	0.1595	-1.6076
4	0.1261	0.0367	-0.1051	0.4293	0.1949	-1.6302
5	0.1267	0.0359	-0.1051	0.4403	0.2217	-1.6454
6	0.1272	0.0351	-0.1051	0.4513	0.2386	-1.6518
7	0.1278	0.0343	-0.1051	0.4621	0.2466	-1.6509
8	0.1284	0.0334	-0.1051	0.4729	0.2455	-1.6428
9	0.1290	0.0326	-0.1051	0.4835	0.2348	-1.6270
10	0.1295	0.0318	-0.1051	0.4940	0.2151	-1.6038
11	0.1267	0.0359	-0.0891	0.5044	0.1863	-1.5726
12	0.1238	0.0400	-0.0811	0.5146	0.1489	-1.5332
13	0.1209	0.0441	-0.0811	0.5248	0.1016	-1.4834
14	0.1181	0.0482	-0.0891	0.5348	0.0448	-1.4224
15	0.1152	0.0523	-0.1051	0.5447	-0.0211	-1.3491
16	0.1158	0.0515	-0.1051	0.5348	-0.0193	-1.3590
17	0.1163	0.0506	-0.1051	0.5248	-0.0176	-1.3691
18	0.1169	0.0498	-0.1051	0.5146	-0.0159	-1.3790
19	0.1175	0.0490	-0.1051	0.5044	-0.0146	-1.3881
20	0.1181	0.0482	-0.1051	0.4940	-0.0132	-1.3971
21	0.1186	0.0474	-0.1051	0.4835	-0.0119	-1.4061
22	0.1192	0.0466	-0.1051	0.4729	-0.0102	-1.4149
23	0.1198	0.0457	-0.1051	0.4621	-0.0096	-1.4231
24	0.1204	0.0449	-0.1051	0.4513	-0.0086	-1.4311
25	0.1209	0.0441	-0.1051	0.4403	-0.0029	-1.4391
26	0.1215	0.0433	-0.1051	0.4293	-0.0066	-1.4463
27	0.1221	0.0425	-0.1051	0.4181	-0.0055	-1.4542
28	0.1227	0.0416	-0.1051	0.4068	-0.0050	-1.4614
29	0.1232	0.0408	-0.1051	0.3954	-0.0043	-1.4679

*Fuente: Elaboración propia.*

## Caminata Cuadrúpeda

Tabla N°3.14. Ángulos calculados por ISim/Caminata Cuadrúpeda

$i$	Posición Calculada (m)			Ángulos Calculados (rad)		
	$x_1$	$y_1$	$z_1$	$\theta_1$	$\theta_2$	$\theta_3$
0	0.1238	0.0400	-0.1051	0.2378	0.0004	-1.5370
1	0.1244	0.0392	-0.1051	0.2444	0.1315	-1.6923
2	0.1249	0.0384	-0.1051	0.2514	0.1973	-1.7432
3	0.1255	0.0375	-0.1051	0.2588	0.2232	-1.7432
4	0.1261	0.0367	-0.1051	0.2668	0.2322	-1.7432
5	0.1267	0.0359	-0.1051	0.2751	0.2222	-1.7432
6	0.1272	0.0351	-0.1051	0.2840	0.1920	-1.7432
7	0.1278	0.0343	-0.1051	0.2934	0.1418	-1.7432
8	0.1284	0.0334	-0.1051	0.3035	0.0753	-1.7432
9	0.1290	0.0326	-0.1051	0.3143	-0.0023	-1.7432
10	0.1295	0.0318	-0.1051	0.3258	-0.0847	-1.7432
11	0.1267	0.0359	-0.0891	0.3199	-0.0735	-1.7432
12	0.1238	0.0400	-0.0811	0.3143	-0.0627	-1.7432
13	0.1209	0.0441	-0.0811	0.3088	-0.0516	-1.7432
14	0.1181	0.0482	-0.0891	0.3035	-0.0407	-1.7432
15	0.1152	0.0523	-0.1051	0.2984	-0.0300	-1.7432
16	0.1158	0.0515	-0.1051	0.2934	-0.0195	-1.7431
17	0.1163	0.0506	-0.1051	0.2886	-0.0165	-1.7302
18	0.1169	0.0498	-0.1051	0.2840	-0.0132	-1.7167
19	0.1175	0.0490	-0.1051	0.2795	-0.0105	-1.7036
20	0.1181	0.0482	-0.1051	0.2751	-0.0082	-1.6896
21	0.1186	0.0474	-0.1051	0.2708	-0.0061	-1.6756
22	0.1192	0.0466	-0.1051	0.2668	-0.0043	-1.6612
23	0.1198	0.0457	-0.1051	0.2627	-0.0029	-1.6469
24	0.1204	0.0449	-0.1051	0.2588	-0.0015	-1.6320
25	0.1209	0.0441	-0.1051	0.2551	-0.0003	-1.6170
26	0.1215	0.0433	-0.1051	0.2514	0.0002	-1.6014
27	0.1221	0.0425	-0.1051	0.2479	0.0006	-1.5856
28	0.1227	0.0416	-0.1051	0.2444	0.0005	-1.5695
29	0.1232	0.0408	-0.1051	0.2410	0.0007	-1.5537

*Fuente: Elaboración propia.*

### Caminata Cuadrúpeda 4+2

Tabla N°3.15. Ángulos calculados por ISim/Caminata Cuadrúpeda 4+2

$i$	Posición Calculada (m)			Ángulos Calculados (rad)		
	$x_1$	$y_1$	$z_1$	$\theta_1$	$\theta_2$	$\theta_3$
0	0.1238	0.0400	-0.1051	0.2378	0.0004	-1.5370
1	0.1244	0.0392	-0.1051	0.2444	0.1315	-1.6923
2	0.1249	0.0384	-0.1051	0.2514	0.1973	-1.7432
3	0.1255	0.0375	-0.1051	0.2588	0.2232	-1.7432
4	0.1261	0.0367	-0.1051	0.2668	0.2322	-1.7432
5	0.1267	0.0359	-0.1051	0.2751	0.2222	-1.7432
6	0.1272	0.0351	-0.1051	0.2840	0.1920	-1.7432
7	0.1278	0.0343	-0.1051	0.2934	0.1418	-1.7432
8	0.1284	0.0334	-0.1051	0.3035	0.0753	-1.7432
9	0.1290	0.0326	-0.1051	0.3143	-0.0023	-1.7432
10	0.1295	0.0318	-0.1051	0.3258	-0.0847	-1.7432
11	0.1267	0.0359	-0.0891	0.3199	-0.0735	-1.7432
12	0.1238	0.0400	-0.0811	0.3143	-0.0627	-1.7432
13	0.1209	0.0441	-0.0811	0.3088	-0.0516	-1.7432
14	0.1181	0.0482	-0.0891	0.3035	-0.0407	-1.7432
15	0.1152	0.0523	-0.1051	0.2984	-0.0300	-1.7432
16	0.1158	0.0515	-0.1051	0.2934	-0.0195	-1.7431
17	0.1163	0.0506	-0.1051	0.2886	-0.0165	-1.7302
18	0.1169	0.0498	-0.1051	0.2840	-0.0132	-1.7167
19	0.1175	0.0490	-0.1051	0.2795	-0.0105	-1.7036
20	0.1181	0.0482	-0.1051	0.2751	-0.0082	-1.6896
21	0.1186	0.0474	-0.1051	0.2708	-0.0061	-1.6756
22	0.1192	0.0466	-0.1051	0.2668	-0.0043	-1.6612
23	0.1198	0.0457	-0.1051	0.2627	-0.0029	-1.6469
24	0.1204	0.0449	-0.1051	0.2588	-0.0015	-1.6320
25	0.1209	0.0441	-0.1051	0.2551	-0.0003	-1.6170
26	0.1215	0.0433	-0.1051	0.2514	0.0002	-1.6014
27	0.1221	0.0425	-0.1051	0.2479	0.0006	-1.5856
28	0.1227	0.0416	-0.1051	0.2444	0.0005	-1.5695
29	0.1232	0.0408	-0.1051	0.2410	0.0007	-1.5537

*Fuente: Elaboración propia.*

## Caminata Pentápoda

Tabla N°3.16. Ángulos calculados por ISim/Caminata Pentápoda

$i$	Posición Calculada (m)			Ángulos Calculados (rad)		
	$x_1$	$y_1$	$z_1$	$\theta_1$	$\theta_2$	$\theta_3$
0	0.1238	0.0400	-0.1051	0.3125	-0.0010	-1.5095
1	0.1244	0.0392	-0.1051	0.3052	-0.0014	-1.5065
2	0.1249	0.0384	-0.1051	0.2979	-0.0013	-1.5040
3	0.1255	0.0375	-0.1051	0.2906	-0.0014	-1.5008
4	0.1261	0.0367	-0.1051	0.2834	-0.0018	-1.4971
5	0.1267	0.0359	-0.1051	0.2762	-0.0021	-1.4939
6	0.1272	0.0351	-0.1051	0.2691	-0.0024	-1.4902
7	0.1278	0.0343	-0.1051	0.2619	-0.0025	-1.4871
8	0.1284	0.0334	-0.1051	0.2548	-0.0032	-1.4833
9	0.1290	0.0326	-0.1051	0.2478	-0.0034	-1.4794
10	0.1295	0.0318	-0.1051	0.2408	-0.0034	-1.4762
11	0.1267	0.0359	-0.0891	0.2762	0.1635	-1.6472
12	0.1238	0.0400	-0.0811	0.3125	0.2483	-1.7304
13	0.1209	0.0441	-0.0811	0.3496	0.2483	-1.7432
14	0.1181	0.0482	-0.0891	0.3876	0.1664	-1.6891
15	0.1152	0.0523	-0.1051	0.4261	0.0005	-1.5454
16	0.1158	0.0515	-0.1051	0.4183	0.0004	-1.5436
17	0.1163	0.0506	-0.1051	0.4106	0.0005	-1.5418
18	0.1169	0.0498	-0.1051	0.4028	0.0004	-1.5394
19	0.1175	0.0490	-0.1051	0.3952	0.0001	-1.5376
20	0.1181	0.0482	-0.1051	0.3876	0.0001	-1.5358
21	0.1186	0.0474	-0.1051	0.3799	0.0000	-1.5333
22	0.1192	0.0466	-0.1051	0.3723	0.0004	-1.5316
23	0.1198	0.0457	-0.1051	0.3648	-0.0003	-1.5285
24	0.1204	0.0449	-0.1051	0.3572	-0.0002	-1.5261
25	0.1209	0.0441	-0.1051	0.3496	-0.0006	-1.5237
26	0.1215	0.0433	-0.1051	0.3422	-0.0002	-1.5213
27	0.1221	0.0425	-0.1051	0.3347	-0.0006	-1.5182
28	0.1227	0.0416	-0.1051	0.3273	-0.0006	-1.5157
29	0.1232	0.0408	-0.1051	0.3199	-0.0009	-1.5127

*Fuente: Elaboración propia.*

### 3.2.5. Técnicas de Procesamiento de datos y análisis de datos

#### a) Cuadro comparativo de la resolución del actuador

Para realizar el análisis de comparación de precisión con los resultados obtenidos es necesario definir la precisión a utilizar. En este trabajo se tomará en cuenta la mayor resolución de los actuadores, lo cual podemos apreciar en la tabla 3.17, cuya mayor resolución contiene 3 decimales.

Tabla N°3.17. Cuadro comparativo de resolución de actuadores

Marca	Tower Pro	Hitec	Dynamixel	Herkulex
Modelo	MG996R	HS-645MG	AX-12	DRS-0201
Resolución	1°	1°	0.29°	0.325°

*Fuente: Elaboración propia.*

#### b) Cuadro comparativo de precisión

A partir de los reportes calculados por el software matemático MATLAB R2015b, el simulador ISim de XILINX ISE DESIGN V14.7, se elabora las tablas 3.18 al 3.21 comparando los resultados de cálculos matemáticos y de los datos obtenidos por la arquitectura propuesta en FPGA. Se ha considerado un error con una precisión de 3 decimales teniendo en consideración el cuadro anterior 3.17 donde se señala dicha precisión como la mayor. Adicionalmente, se puede observar que los ángulos obtenidos han sido utilizados para calcular las posiciones con la cinemática directa (FK) y verificar el error.



### Caminata Trípode

Tabla N°3.18. Cuadro comparativo de los ángulos calculados por Matlab y el simulador  
ISim/Caminata Trípode

i	Matlab (rad)			ISim (rad)			Error Relativo Porcentual %			FK-Matlab (m)			FK-Isim (m)		
	$\theta_1$	$\theta_2$	$\theta_3$	$\theta_1$	$\theta_2$	$\theta_3$	$\theta_1$	$\theta_2$	$\theta_3$	$x_1$	$y_1$	$z_1$	$x_1$	$y_1$	$z_1$
0	0.384	-0.005	-1.474	0.384	0.000	-1.474	0.000	100.000	0.000	0.124	0.050	-0.105	0.124	0.050	-0.105
1	0.395	0.059	-1.53	0.395	0.060	-1.530	0.000	1.695	0.000	0.124	0.052	-0.099	0.124	0.052	-0.099
2	0.407	0.114	-1.574	0.407	0.115	-1.575	0.000	0.877	0.064	0.124	0.053	-0.094	0.124	0.053	-0.094
3	0.418	0.159	-1.607	0.418	0.159	-1.608	0.000	0.000	0.062	0.124	0.055	-0.089	0.124	0.055	-0.089
4	0.429	0.194	-1.63	0.429	0.195	-1.63	0.000	0.515	0.000	0.124	0.057	-0.086	0.124	0.057	-0.086
5	0.44	0.221	-1.645	0.44	0.222	-1.645	0.000	0.452	0.000	0.124	0.058	-0.083	0.124	0.058	-0.083
6	0.451	0.238	-1.651	0.451	0.239	-1.652	0.000	0.420	0.061	0.124	0.060	-0.081	0.124	0.060	-0.081
7	0.462	0.246	-1.65	0.462	0.247	-1.651	0.000	0.407	0.061	0.124	0.062	-0.080	0.124	0.062	-0.080
8	0.473	0.245	-1.642	0.473	0.245	-1.643	0.000	0.000	0.061	0.124	0.063	-0.080	0.124	0.063	-0.080
9	0.483	0.234	-1.626	0.483	0.235	-1.627	0.000	0.427	0.062	0.124	0.065	-0.081	0.124	0.065	-0.081
10	0.494	0.214	-1.603	0.494	0.215	-1.604	0.000	0.467	0.062	0.124	0.067	-0.083	0.124	0.067	-0.083
11	0.504	0.186	-1.572	0.504	0.186	-1.573	0.000	0.000	0.064	0.124	0.068	-0.086	0.124	0.068	-0.086
12	0.515	0.148	-1.532	0.515	0.149	-1.533	0.000	0.676	0.065	0.124	0.070	-0.089	0.124	0.070	-0.089
13	0.525	0.101	-1.483	0.525	0.102	-1.483	0.000	0.990	0.000	0.124	0.072	-0.094	0.124	0.072	-0.094
14	0.535	0.044	-1.422	0.535	0.045	-1.422	0.000	2.273	0.000	0.124	0.073	-0.099	0.124	0.073	-0.099
15	0.545	-0.022	-1.348	0.545	-0.021	-1.349	0.000	4.545	0.074	0.124	0.075	-0.105	0.124	0.075	-0.105
16	0.535	-0.02	-1.358	0.535	-0.019	-1.359	0.000	5.000	0.074	0.124	0.073	-0.105	0.124	0.073	-0.105
17	0.525	-0.018	-1.368	0.525	-0.018	-1.369	0.000	0.000	0.073	0.124	0.072	-0.105	0.124	0.072	-0.105
18	0.515	-0.017	-1.378	0.515	-0.016	-1.379	0.000	5.882	0.073	0.124	0.070	-0.105	0.124	0.070	-0.105
19	0.504	-0.015	-1.387	0.504	-0.015	-1.388	0.000	0.000	0.072	0.124	0.068	-0.105	0.124	0.068	-0.105
20	0.494	-0.014	-1.396	0.494	-0.013	-1.397	0.000	7.143	0.072	0.124	0.067	-0.105	0.124	0.067	-0.105
21	0.483	-0.013	-1.405	0.483	-0.012	-1.406	0.000	7.692	0.071	0.124	0.065	-0.105	0.124	0.065	-0.105
22	0.473	-0.012	-1.413	0.473	-0.01	-1.415	0.000	16.667	0.142	0.124	0.063	-0.105	0.124	0.063	-0.105
23	0.462	-0.01	-1.422	0.462	-0.01	-1.423	0.000	0.000	0.070	0.124	0.062	-0.105	0.124	0.062	-0.105
24	0.451	-0.009	-1.43	0.451	-0.009	-1.431	0.000	0.000	0.070	0.124	0.060	-0.105	0.124	0.060	-0.105
25	0.44	-0.008	-1.438	0.44	-0.003	-1.439	0.000	62.500	0.070	0.124	0.058	-0.105	0.124	0.058	-0.105
26	0.429	-0.008	-1.445	0.429	-0.007	-1.446	0.000	12.500	0.069	0.124	0.057	-0.105	0.124	0.057	-0.105
27	0.418	-0.007	-1.453	0.418	-0.005	-1.454	0.000	28.571	0.069	0.124	0.055	-0.105	0.124	0.055	-0.105
28	0.407	-0.006	-1.46	0.407	-0.005	-1.461	0.000	16.667	0.068	0.124	0.053	-0.105	0.124	0.053	-0.105
29	0.395	-0.005	-1.467	0.395	-0.004	-1.468	0.000	20.000	0.068	0.124	0.052	-0.105	0.124	0.052	-0.105

Fuente: Elaboración propia.

## Caminata Cuadrúpeda

Tabla N°3.19. Cuadro comparativo entre ángulos calculados por Matlab y el simulador  
ISim/Caminata Cuadrúpeda

i	Matlab (rad)			ISim (rad)			Error Relativo Porcentual %			FK-Matlab (mm)			FK-Isim (mm)		
	$\theta_1$	$\theta_2$	$\theta_3$	$\theta_1$	$\theta_2$	$\theta_3$	$\theta_1$	$\theta_2$	$\theta_3$	$x_1$	$y_1$	$z_1$	$x_1$	$y_1$	$z_1$
0	0.238	-0.001	-1.536	0.238	0.000	-1.537	0.000	100.000	0.065	0.124	0.030	-0.105	0.124	0.030	-0.105
1	0.244	0.131	-1.692	0.244	0.132	-1.692	0.000	0.763	0.000	0.120	0.030	-0.093	0.120	0.030	-0.093
2	0.251	0.235	-1.811	0.251	0.197	-1.743	0.000	16.170	3.755	0.117	0.030	-0.083	0.121	0.031	-0.086
3	0.259	0.310	-1.900	0.259	0.223	-1.743	0.000	28.065	8.263	0.113	0.030	-0.076	0.123	0.032	-0.084
4	0.267	0.356	-1.964	0.267	0.232	-1.743	0.000	34.831	11.253	0.110	0.030	-0.072	0.123	0.034	-0.083
5	0.275	0.369	-2.005	0.275	0.222	-1.743	0.000	39.837	13.067	0.106	0.030	-0.070	0.122	0.034	-0.084
6	0.284	0.350	-2.025	0.284	0.192	-1.743	0.000	45.143	13.926	0.103	0.030	-0.072	0.119	0.035	-0.087
7	0.293	0.297	-2.019	0.293	0.142	-1.743	0.000	52.189	13.670	0.099	0.030	-0.076	0.115	0.035	-0.091
8	0.303	0.212	-1.981	0.303	0.075	-1.743	0.000	64.623	12.014	0.096	0.030	-0.083	0.108	0.034	-0.097
9	0.314	0.096	-1.920	0.314	-0.002	-1.743	0.000	102.083	9.219	0.092	0.030	-0.093	0.100	0.033	-0.104
10	0.326	-0.045	-1.815	0.326	-0.085	-1.743	0.000	88.889	3.967	0.089	0.030	-0.105	0.092	0.031	-0.110
11	0.320	-0.040	-1.803	0.320	-0.073	-1.743	0.000	82.500	3.328	0.091	0.030	-0.105	0.093	0.031	-0.109
12	0.314	-0.036	-1.792	0.314	-0.063	-1.743	0.000	75.000	2.734	0.092	0.030	-0.105	0.094	0.031	-0.108
13	0.309	-0.031	-1.780	0.309	-0.052	-1.743	0.000	67.742	2.079	0.094	0.030	-0.105	0.096	0.031	-0.107
14	0.303	-0.027	-1.768	0.303	-0.041	-1.743	0.000	51.852	1.414	0.096	0.030	-0.105	0.097	0.030	-0.107
15	0.298	-0.024	-1.755	0.298	-0.030	-1.743	0.000	25.000	0.684	0.098	0.030	-0.105	0.098	0.030	-0.106
16	0.293	-0.020	-1.743	0.293	-0.020	-1.743	0.000	0.000	0.000	0.099	0.030	-0.105	0.099	0.030	-0.105
17	0.288	-0.017	-1.729	0.289	-0.016	-1.730	0.347	5.882	0.058	0.101	0.030	-0.105	0.101	0.030	-0.105
18	0.284	-0.014	-1.717	0.284	-0.013	-1.717	0.000	7.143	0.000	0.103	0.030	-0.105	0.103	0.030	-0.105
19	0.279	-0.011	-1.703	0.279	-0.011	-1.704	0.000	0.000	0.059	0.105	0.030	-0.105	0.105	0.030	-0.105
20	0.275	-0.009	-1.689	0.275	-0.008	-1.690	0.000	11.111	0.059	0.106	0.030	-0.105	0.106	0.030	-0.105
21	0.271	-0.007	-1.675	0.271	-0.006	-1.676	0.000	14.286	0.060	0.108	0.030	-0.105	0.108	0.030	-0.105
22	0.267	-0.005	-1.661	0.267	-0.004	-1.661	0.000	20.000	0.000	0.110	0.030	-0.105	0.110	0.030	-0.105
23	0.263	-0.003	-1.646	0.263	-0.003	-1.647	0.000	0.000	0.061	0.112	0.030	-0.105	0.112	0.030	-0.105
24	0.259	-0.002	-1.631	0.259	-0.001	-1.632	0.000	50.000	0.061	0.113	0.030	-0.105	0.113	0.030	-0.105
25	0.255	-0.001	-1.616	0.255	0.000	-1.617	0.000	100.000	0.062	0.115	0.030	-0.105	0.115	0.030	-0.105
26	0.251	-0.001	-1.601	0.251	0.000	-1.601	0.000	100.000	0.000	0.117	0.030	-0.105	0.117	0.030	-0.105
27	0.248	0.000	-1.585	0.248	0.000	-1.586	0.000	0.000	0.063	0.119	0.030	-0.105	0.119	0.030	-0.105
28	0.244	0.000	-1.569	0.244	0.000	-1.569	0.000	0.000	0.000	0.120	0.030	-0.105	0.120	0.030	-0.105
29	0.241	0.000	-1.552	0.241	0.000	-1.554	0.000	0.000	0.129	0.122	0.030	-0.105	0.122	0.030	-0.105

*Fuente: Elaboración propia.*

### Caminata Cuadrúpeda 4+2

Tabla N°3.20. Cuadro comparativo entre ángulos calculados por Matlab y el simulador  
ISim/Caminata Cuadrúpeda 4+2

i	Matlab (rad)			ISim (rad)			Error Relativo Porcentual %			FK-Matlab (mm)			FK-Isim (mm)		
	$\theta_1$	$\theta_2$	$\theta_3$	$\theta_1$	$\theta_2$	$\theta_3$	$\theta_1$	$\theta_2$	$\theta_3$	$x_1$	$y_1$	$z_1$	$x_1$	$y_1$	$z_1$
0	0.238	-0.001	-1.536	0.238	0.000	-1.537	0.000	100.000	0.065	0.124	0.030	-0.105	0.124	0.030	-0.105
1	0.244	0.131	-1.692	0.244	0.132	-1.692	0.000	0.763	0.000	0.120	0.030	-0.093	0.120	0.030	-0.093
2	0.251	0.235	-1.811	0.251	0.197	-1.743	0.000	16.170	3.755	0.117	0.030	-0.083	0.121	0.031	-0.086
3	0.259	0.310	-1.900	0.259	0.223	-1.743	0.000	28.065	8.263	0.113	0.030	-0.076	0.123	0.032	-0.084
4	0.267	0.356	-1.964	0.267	0.232	-1.743	0.000	34.831	11.253	0.110	0.030	-0.072	0.123	0.034	-0.083
5	0.275	0.369	-2.005	0.275	0.222	-1.743	0.000	39.837	13.067	0.106	0.030	-0.070	0.122	0.034	-0.084
6	0.284	0.350	-2.025	0.284	0.192	-1.743	0.000	45.143	13.926	0.103	0.030	-0.072	0.119	0.035	-0.087
7	0.293	0.297	-2.019	0.293	0.142	-1.743	0.000	52.189	13.670	0.099	0.030	-0.076	0.115	0.035	-0.091
8	0.303	0.212	-1.981	0.303	0.075	-1.743	0.000	64.623	12.014	0.096	0.030	-0.083	0.108	0.034	-0.097
9	0.314	0.096	-1.920	0.314	-0.002	-1.743	0.000	102.083	9.219	0.092	0.030	-0.093	0.100	0.033	-0.104
10	0.326	-0.045	-1.815	0.326	-0.085	-1.743	0.000	88.889	3.967	0.089	0.030	-0.105	0.092	0.031	-0.110
11	0.320	-0.040	-1.803	0.320	-0.073	-1.743	0.000	82.500	3.328	0.091	0.030	-0.105	0.093	0.031	-0.109
12	0.314	-0.036	-1.792	0.314	-0.063	-1.743	0.000	75.000	2.734	0.092	0.030	-0.105	0.094	0.031	-0.108
13	0.309	-0.031	-1.780	0.309	-0.052	-1.743	0.000	67.742	2.079	0.094	0.030	-0.105	0.096	0.031	-0.107
14	0.303	-0.027	-1.768	0.303	-0.041	-1.743	0.000	51.852	1.414	0.096	0.030	-0.105	0.097	0.030	-0.107
15	0.298	-0.024	-1.755	0.298	-0.030	-1.743	0.000	25.000	0.684	0.098	0.030	-0.105	0.098	0.030	-0.106
16	0.293	-0.020	-1.743	0.293	-0.020	-1.743	0.000	0.000	0.000	0.099	0.030	-0.105	0.099	0.030	-0.105
17	0.288	-0.017	-1.729	0.289	-0.016	-1.730	0.347	5.882	0.058	0.101	0.030	-0.105	0.101	0.030	-0.105
18	0.284	-0.014	-1.717	0.284	-0.013	-1.717	0.000	7.143	0.000	0.103	0.030	-0.105	0.103	0.030	-0.105
19	0.279	-0.011	-1.703	0.279	-0.011	-1.704	0.000	0.000	0.059	0.105	0.030	-0.105	0.105	0.030	-0.105
20	0.275	-0.009	-1.689	0.275	-0.008	-1.690	0.000	11.111	0.059	0.106	0.030	-0.105	0.106	0.030	-0.105
21	0.271	-0.007	-1.675	0.271	-0.006	-1.676	0.000	14.286	0.060	0.108	0.030	-0.105	0.108	0.030	-0.105
22	0.267	-0.005	-1.661	0.267	-0.004	-1.661	0.000	20.000	0.000	0.110	0.030	-0.105	0.110	0.030	-0.105
23	0.263	-0.003	-1.646	0.263	-0.003	-1.647	0.000	0.000	0.061	0.112	0.030	-0.105	0.112	0.030	-0.105
24	0.259	-0.002	-1.631	0.259	-0.001	-1.632	0.000	50.000	0.061	0.113	0.030	-0.105	0.113	0.030	-0.105
25	0.255	-0.001	-1.616	0.255	0.000	-1.617	0.000	100.000	0.062	0.115	0.030	-0.105	0.115	0.030	-0.105
26	0.251	-0.001	-1.601	0.251	0.000	-1.601	0.000	100.000	0.000	0.117	0.030	-0.105	0.117	0.030	-0.105
27	0.248	0.000	-1.585	0.248	0.000	-1.586	0.000	0.000	0.063	0.119	0.030	-0.105	0.119	0.030	-0.105
28	0.244	0.000	-1.569	0.244	0.000	-1.569	0.000	0.000	0.000	0.120	0.030	-0.105	0.120	0.030	-0.105
29	0.241	0.000	-1.552	0.241	0.000	-1.554	0.000	0.000	0.129	0.122	0.030	-0.105	0.122	0.030	-0.105

Fuente: Elaboración propia.

## Caminata Pentápoda

Tabla N°3.21. Cuadro comparativo entre ángulos calculados por Matlab y el simulador

ISim/Caminata Pentápoda

i	Matlab (rad)			ISim (rad)			Error Relativo Porcentual %			FK-Matlab (mm)			FK-Isim (mm)		
	$\theta_1$	$\theta_2$	$\theta_3$	$\theta_1$	$\theta_2$	$\theta_3$	$\theta_1$	$\theta_2$	$\theta_3$	$x_1$	$y_1$	$z_1$	$x_1$	$y_1$	$z_1$
0	0.313	-0.002	-1.509	0.313	-0.001	-1.51	0.000	50.000	0.066	0.124	0.04	-0.105	0.124	0.04	-0.105
1	0.305	-0.002	-1.506	0.305	-0.001	-1.506	0.000	50.000	0.000	0.124	0.039	-0.105	0.124	0.039	-0.105
2	0.298	-0.002	-1.503	0.298	-0.001	-1.504	0.000	50.000	0.067	0.125	0.038	-0.105	0.125	0.038	-0.105
3	0.291	-0.003	-1.5	0.291	-0.001	-1.501	0.000	66.667	0.067	0.126	0.038	-0.105	0.126	0.038	-0.105
4	0.283	-0.003	-1.496	0.283	-0.002	-1.497	0.000	33.333	0.067	0.126	0.037	-0.105	0.126	0.037	-0.105
5	0.276	-0.003	-1.493	0.276	-0.002	-1.494	0.000	33.333	0.067	0.127	0.036	-0.105	0.127	0.036	-0.105
6	0.269	-0.003	-1.49	0.269	-0.002	-1.49	0.000	33.333	0.000	0.127	0.035	-0.105	0.127	0.035	-0.105
7	0.262	-0.004	-1.486	0.262	-0.003	-1.487	0.000	25.000	0.067	0.128	0.034	-0.105	0.128	0.034	-0.105
8	0.262	-0.004	-1.48	0.255	-0.003	-1.483	2.672	25.000	0.203	0.128	0.033	-0.105	0.128	0.033	-0.105
9	0.248	-0.004	-1.478	0.248	-0.003	-1.479	0.000	25.000	0.068	0.129	0.033	-0.105	0.129	0.033	-0.105
10	0.241	-0.005	-1.475	0.241	-0.003	-1.476	0.000	40.000	0.068	0.13	0.032	-0.105	0.13	0.032	-0.105
11	0.276	0.163	-1.646	0.276	0.164	-1.647	0.000	0.613	0.061	0.127	0.036	-0.089	0.127	0.036	-0.089
12	0.313	0.248	-1.73	0.313	0.248	-1.73	0.000	0.000	0.000	0.124	0.04	-0.081	0.124	0.04	-0.081
13	0.35	0.249	-1.745	0.35	0.248	-1.743	0.000	0.402	0.115	0.121	0.044	-0.081	0.121	0.044	-0.081
14	0.387	0.166	-1.688	0.388	0.166	-1.689	0.258	0.000	0.059	0.118	0.048	-0.089	0.118	0.048	-0.089
15	0.426	0.000	-1.545	0.426	0.000	-1.545	0.000	0.000	0.000	0.115	0.052	-0.105	0.115	0.052	-0.105
16	0.418	0.000	-1.543	0.418	0.000	-1.544	0.000	0.000	0.065	0.116	0.051	-0.105	0.116	0.051	-0.105
17	0.411	0.000	-1.541	0.411	0.000	-1.542	0.000	0.000	0.065	0.116	0.051	-0.105	0.116	0.051	-0.105
18	0.403	-0.001	-1.539	0.403	0.000	-1.539	0.000	100.000	0.000	0.117	0.05	-0.105	0.117	0.05	-0.105
19	0.395	-0.001	-1.537	0.395	0.000	-1.538	0.000	100.000	0.065	0.117	0.049	-0.105	0.117	0.049	-0.105
20	0.387	-0.001	-1.534	0.388	0.000	-1.536	0.258	100.000	0.130	0.118	0.048	-0.105	0.118	0.048	-0.105
21	0.380	-0.001	-1.533	0.38	0.000	-1.533	0.000	100.000	0.000	0.119	0.047	-0.105	0.119	0.047	-0.105
22	0.372	-0.001	-1.53	0.372	0.000	-1.532	0.000	100.000	0.131	0.119	0.047	-0.105	0.119	0.047	-0.105
23	0.365	-0.001	-1.528	0.365	0.000	-1.529	0.000	100.000	0.065	0.12	0.046	-0.105	0.12	0.046	-0.105
24	0.357	-0.001	-1.525	0.357	0.000	-1.526	0.000	100.000	0.066	0.12	0.045	-0.105	0.12	0.045	-0.105
25	0.350	-0.001	-1.523	0.350	-0.001	-1.524	0.000	0.000	0.066	0.121	0.044	-0.105	0.121	0.044	-0.105
26	0.342	-0.001	-1.52	0.342	0.000	-1.521	0.000	100.000	0.066	0.122	0.043	-0.105	0.122	0.043	-0.105
27	0.335	-0.001	-1.517	0.335	-0.001	-1.518	0.000	0.000	0.066	0.122	0.042	-0.105	0.122	0.042	-0.105
28	0.327	-0.002	-1.514	0.327	-0.001	-1.516	0.000	50.000	0.132	0.123	0.042	-0.105	0.123	0.042	-0.105
29	0.32	-0.002	-1.512	0.32	-0.001	-1.513	0.000	50.000	0.066	0.123	0.041	-0.105	0.123	0.041	-0.105

*Fuente: Elaboración propia.*

**c) Análisis de los recursos de hardware de la entidad obtenida**

De la arquitectura de la entidad propuesta, se obtiene el siguiente cuadro de los recursos de hardware utilizados, en los cuales podemos observar la diferencia entre el hardware utilizado por la arquitectura respecto al modelo de FPGA elegido para la simulación.

Sintetizando el programa de ISE Project (anexo 2 al 6), donde se utiliza como muestra el dispositivo FPGA xc3s500e-4pq208 como lo indicado en puntos anteriores.

Tabla N°3.22. Recursos utilizados por el dispositivo FPGA xc3s500e-4pq208

Resumen utilizado de hardware (valores estimados)			
Hardware utilizado	Usado	Disponible	Utilizado
Slices	14098	4656	302%
Slice Flip Flops	2045	9312	21%
4 input LUTs	27072	9312	290%
Bonded IOBs	198	158	125%
MULT18X18SIOs	4	20	20%
GCLKs	1	24	4%

*Fuente: ISE Design Summary.*

De la tabla 3.22 se rescata la cantidad de hardware usado por la arquitectura planteada, con la cual se debe realizar la selección correcta del dispositivo FPGA a utilizar. Se debe tener en cuenta que la arquitectura planteada es una entidad la cual irá en un SOC como parte de una arquitectura más compleja para alguna aplicación robótica, por lo cual el número de IOBs mostrados son señales internas que solo por facilidad de simulación se realizaron como señales físicas.

# **CAPITULO IV**

## 4. RESULTADOS

### Ecuaciones Cinemática Inversa para el Algoritmo CORDIC

Se adaptaron las ecuaciones de cinemática inversa de una extremidad robótica hexápoda de 3DOF de tal manera que puedan ser implementadas utilizando el algoritmo CORDIC, como se muestra a continuación:

$$\theta_1 = \text{atan} (y_e/x_e) \quad (4.1)$$

$$\theta_2 = \text{atan} \left( \frac{G}{\sqrt{1-G^2}} \right) - \text{atan} \left( \frac{\sin \theta_3}{F+\cos \theta_3} \right) \quad (4.2)$$

$$\theta_3 = \text{atan} \left( \frac{\sqrt{1-D^2}}{D} \right) \quad (4.3)$$

Variables:

$$r = \sqrt{x_e^2 + y_e^2} \quad (4.4)$$

$$A = 2l_1 r \quad (4.5)$$

$$B = \sqrt{(r - l_1)^2 + z_e^2} \quad (4.6)$$

$$C = r^2 + z_e^2 + l_1^2 - l_2^2 - l_3^2 - A \quad (4.7)$$

$$D = C C_a = \cos \theta_3 \quad (4.8)$$

$$G = \frac{z_e}{B} \quad (4.9)$$

Parámetros:

$$C_a = \frac{1}{2l_2l_3} \quad (4.10)$$

$$F = \frac{l_2}{l_3} \quad (4.11)$$

## Operadores CORDIC para la cinemática inversa

La tabla 4.1 muestra los operadores CORDIC utilizados para el desarrollo de la cinemática inversa, los cuales resuelven las ecuaciones 4.1, 4.2 y 4.3.

Tabla N°4.1. Operadores CORDIC

Circular Rotacional (CR)	Circular Vectorial (CV)	Hiperbólico Vectorial (HV)
$x \rightarrow$ $y \rightarrow$ $z \rightarrow$ <div style="display: inline-block; border: 1px solid black; padding: 2px; margin: 5px;">CR</div> $\rightarrow K(x \cos z - y \sin z)$ $\rightarrow K(y \cos z + x \sin z)$ $\rightarrow 0$	$x \rightarrow$ $y \rightarrow$ $z \rightarrow$ <div style="display: inline-block; border: 1px solid black; padding: 2px; margin: 5px;">CV</div> $\rightarrow K\sqrt{x^2 + y^2}$ $\rightarrow 0$ $\rightarrow z + \tan^{-1}(y/x)$	$x \rightarrow$ $y \rightarrow$ $z \rightarrow$ <div style="display: inline-block; border: 1px solid black; padding: 2px; margin: 5px;">HV</div> $\rightarrow K'\sqrt{x^2 - y^2}$ $\rightarrow 0$ $\rightarrow z + \tanh^{-1}(y/x)$

*Fuente: Elaboración Propia*

## Entidad CORDIC

En la figura 4.1 se muestra la entidad diseñada para el algoritmo CORDIC.

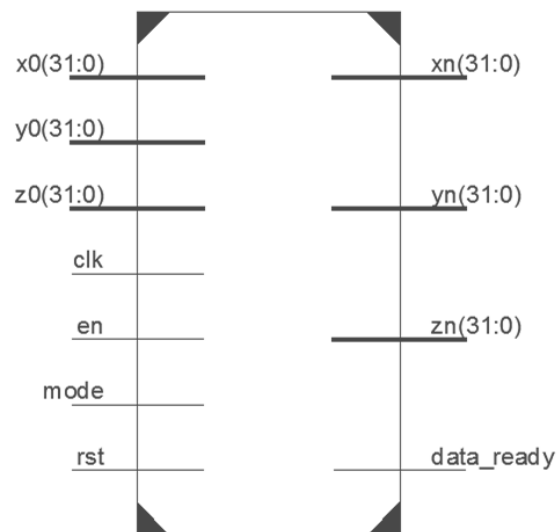


Figura N°4.1. Diseño de entidad propuesta

*Fuente: Elaboración Propia*

ENTRADAS	$x_0, y_0, z_0$ (Vector 32 bits): Parámetros de entrada.
	Clk (Booleano): Señal de reloj.
	En (Booleano): Habilitador.
	Mode (Booleano): Selección de modo de operación.
	Rst (Booleano): Reset.



SALIDAS	{	$x_n, y_n, z_n$ (Vector 32 bits): Parámetros de salida.
		Data_ready (Booleano): Indica cuando los parámetros de salida son datos válidos.

### Arquitectura para el cálculo de la cinemática inversa

En la figura 4.2 se muestra la arquitectura correspondiente a la entidad de la figura 4.1, la cual desarrolla el cálculo cinemático inverso de la extremidad robótica hexápoda implementable en FPGA.

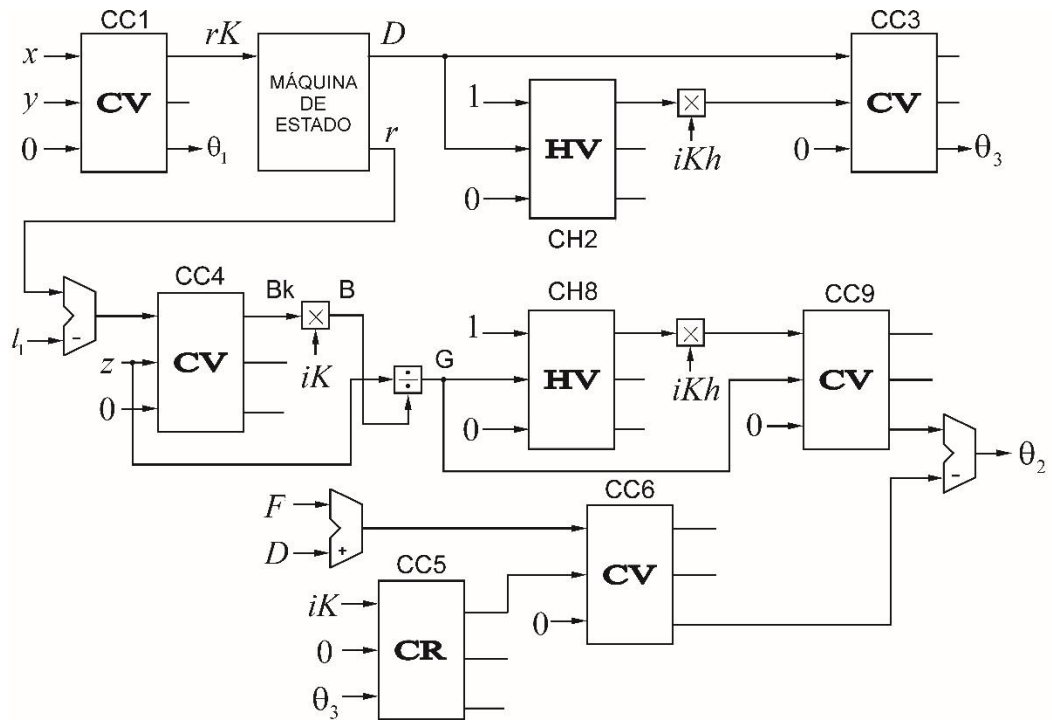


Figura N°4.2. Arquitectura para calcular la cinemática inversa

*Fuente: Elaboración propia*

### Implementación de la arquitectura en FPGA mediante el simulador ISim

De las tablas comparación entre Matlab y el simulador ISim presentadas en el apartado 3.2.5 (tablas 3.18 al 3.21), se observa que existe un error relativo porcentual mínimo es de 0% para los ángulos Q1, Q2 y Q3 como es indicado en la tabla 4.2 y un error relativo porcentual máximo de 2.672% para Q1, 102.083% para Q2 y 13.926% para Q3 en las cuatro caminatas, teniendo a su vez un error relativo porcentual promedio de 0% a 0.102% en Q1, de 7.212% a 44.423% en Q2 y de 0.057% a 3.335% para Q3.

Tabla 4.2: Error relativo porcentual según tipo de caminata

Tipo de Caminata	Error Relativo Porcentual (%)						Error Relativo Porcentual Promedio (%)		
	$\theta_1$		$\theta_2$		$\theta_3$		$\theta_1$	$\theta_2$	$\theta_3$
	Mín	Máx	Mín	Máx	Mín	Máx			
Trípode	0	0	0	62.5	0	0.142	0	7.412	0.057
Cuadrúpeda	0	0.347	0	102.083	0	13.926	0.012	39.437	3.335
Cuadrúpeda 4+2	0	0.347	0	102.083	0	13.926	0.012	39.437	3.335
Pentápoda	0	2.672	0	100	0	0.203	0.106	44.423	0.065

*Fuente: Elaboración Propia.*

### Recursos de hardware del FPGA

Se puede apreciar en la tabla 4.3 los recursos de hardware resultante de la síntesis de la arquitectura propuesta en el presente trabajo.

Tabla 4.3: Recursos utilizados por arquitectura propuesta

Hardware utilizado (valores estimados)	
Hardware	Cantidad
Slices	14098
Slice Flip Flops	2045
4 input LUTs	27072
Bonded IOBs	198
MULT18X18SIOs	4
GCLKs	1

*Fuente: Elaboración Propia.*

# **CAPITULO V**

## 5. DISCUSIÓN DE RESULTADOS

De las ecuaciones 4.1 a 4.3, se puede apreciar que fueron adaptadas de tal manera que sean resueltas por los operadores CORDIC diseñados. Se aprecia también, que la precisión del cálculo en el segundo ángulo es afectada por la precisión en el cálculo del tercero.

En los resultados de la tabla 4.1 se presentan los operadores CORDIC utilizados en el presente trabajo. Sin embargo, las operaciones de multiplicación y división fueron realizadas por los multiplicadores por hardware del FPGA seleccionado, pudiendo estos ser resueltos por una entidad CORDIC en su modo lineal siendo este un criterio de diseño.

De la figura 4.1, podemos observar la entidad obtenida de la arquitectura propuesta para el cálculo cinemático inverso, la cual es un bloque con sus respectivas señales de entrada (posición del elemento final, señal de reloj, habilitadores) y sus salidas (ángulos cinemáticos). En el presente trabajo, se considera una señal de salida Data\_ready para lectura de un dato válido para cada ángulo. Sin embargo, según los requerimientos para el diseño del dispositivo SoC donde la arquitectura puede ser implementada, esto puede ser reemplazado por una sola señal para lectura por flancos de todos los ángulos.

De la figura 4.2, se obtiene la arquitectura para calcular la cinemática inversa de la extremidad de un robot hexápodo. En la cual se puede observar los bloques de los operadores de CORDIC y una máquina de estado para resolver operaciones repetitivas. En la arquitectura propuesta existen entidades de operadores CORDIC que fueron instanciadas varias veces a pesar de que estas son idénticas y suponen un incremento de los recursos utilizados. Esto podría ser mejorado, diseñando un mecanismo para compartir los recursos entre las entidades así poder reutilizarlas en el cálculo. Esto implicaría el aumento del tiempo de procesamiento de la arquitectura, puesto que el cálculo de las ecuaciones dejaría de ser concurrente.

De los resultados presentados en la tabla 4.2 se puede apreciar que el valor del error relativo porcentual promedio es superior al 10% para el ángulo Q2, a la vez el error máximo tiende a valores superiores al 100% para el mismo ángulo y mayor al 10% para Q3. Considerando las tablas 3.17 al 3.20 se aprecia que Q2 tiende constantemente a cero para la caminata

cuadrúpeda y pentápoda, y sucede lo mismo para las caminatas trípode y cuadrúpeda 4+2 pero por tramos. Por otro lado, Q3 tiene a cero sólo en determinadas iteraciones en todas las caminatas. Sin embargo, si apreciamos de la tabla 3.18 a la 3.21, estos errores se vuelven despreciables cuando se hace la cinemática directa para el cálculo de la posición del efector final de la rebotica hexápoda.

De los resultados de la tabla 4.3, se pueden apreciar los recursos de hardware utilizados por la arquitectura propuesta en la presente investigación. Se observa un alto número de Bonded IOBs, los cuales corresponden a los tres *inputs*,  $x_0$ ,  $y_0$  y  $z_0$ , así como a tres *outputs*, Q1, Q2 y Q3, de 32 bits en punto flotante. Donde tres *inputs* son para habilitación, reloj y reset de la entidad, además tres *outputs* para señalización por flancos para cada ángulo de salida, los cuales son de tipo binario. Esto es debido a que los *inputs/output* fueron instanciados en pines físicos para fines de simulación y el alcance del presente trabajo; sin embargo, la arquitectura presentada está proyectada como parte de un SoC para lo cual no se utilizaría ningún recurso Bonded IOBs. Por otro lado, los recursos de Slices y 4 input LUTs corresponden a los recursos utilizados por la máquina de estados que controla la arquitectura y las ocho entidades CORDIC utilizadas, esto puede ser reducido en un posterior trabajo con un mecanismo para compartir recursos de entidades CORDIC, también se utilizaron cuatro multiplicadores por hardware embebidos para las distintas operaciones de la arquitectura y un GCLK para la señalización del reloj. Por último, las entidades CORDIC fueron diseñadas con un número fijo de iteraciones como criterio de diseño tomando como referencia los trabajos citados, estas mantendrán una precisión fija aceptable. Sin embargo, según los requerimientos de hardware, este número de iteraciones puede ser modificado afectando la precisión en el cálculo de los ángulos cinemáticos inversos.

# **CAPITULO VI**

## 6. CONCLUSIONES

- Se logró adaptar las ecuaciones de la cinemática inversa para obtener un modelo implementable en VHDL, utilizando el algoritmo CORDIC.
- Se logró realizar un análisis del algoritmo de CORDIC, realizando un resumen de las arquitecturas, los métodos para implementar el algoritmo y las consideraciones necesarias según el desarrollo previo del modelo matemático.
- En el diseño de la entidad, se logró realizar el diseño apropiado para ser modelado en VHDL considerando los parámetros de entrada y salidas según el modelo matemático de la cinemática inversa.
- Se logró realizar el diseño de la arquitectura y la simulación en el simulador ISim mediante VHDL, demostrando con los resultados que se pudo obtener los ángulos de articulación de la cinemática inversa con errores mínimos dentro un margen considerable. Además, se logró estimar la cantidad de recursos utilizados por la arquitectura propuesta a fin de poder dimensionar el dispositivo adecuado para la implementación de la arquitectura.



# **CAPITULO VII**

## **7. RECOMENDACIONES**

- Para optimizar el número de entidades de CORDIC utilizadas en la arquitectura, se debe realizar un análisis de tiempo. Así, utilizando técnicas para compartir los recursos CORDIC, podría disminuirse la cantidad de entidades utilizadas incluso a una sola, si bien es cierto aumentaría el tiempo de procesamiento, pero optimizaría los recursos utilizados para fines de implementación en hardware.
- El tiempo estimado para resolver los ángulos de la cinemática teóricamente deben ser cumplidos, esto es debido, a que se ha diseñado sumadores y multiplicadores de 32 bits, para lo cual se debe realizar una implementación en hardware para su verificación de los dichos tiempos en una aplicación real.

# **CAPITULO VIII**

## **8. REFERENCIAS BIBLIOGRÁFICAS**

### **LIBROS**

*Siegwart & I.Nourbakhsh (2004), "Autonomous Mobile Robots".*

*J.Craig (2006), "Robótica".*

*P.Chu (2008), "FPGA Prototyping by VHDL Examples".*

*D. Maxinez (2014), "Programación de Sistemas Digitales con VHDL".*

*P. Chu (2006), "RTL Hardware Design using VHDL".*

### **TESIS**

*N.Martinez y P.Fernández (2003), "Robot Hexápodo".*

*G.Cuaya (2007), "Procesos de Decisión de Markov Aplicados a la Locomoción de Robots Hexápodos".*

*R. Schweers (2002), "Descripción en VHDL de arquitecturas para implementar el algoritmo CORDIC".*

*C.Agurto (2006), "Implementación de Arquitecturas para el cálculo de funciones trascendentales empleando el Algoritmo CORDIC en un FPGA".*

*G.Evangelista (2013), "Desarrollo y construcción de una plataforma de investigación robótica hexápoda mediante el uso de un dispositivo MBED NXP LPC1768".*

### **PUBLICACIONES**

*M. García, E. Gorrostieta, E. Vargas, J.Ramos, A.Sotomayor, J.Moya (2012), "Kinematics analysis for trajectory generation in one leg of a hexapod robot".*

- P.Hodilmes, R.Full, D. Koditschek, J.Guckenheimer (2006), "The Dynamics of Legged Locomotion: Models, Analyses, and Challenges".*
- M.Billah, M. Ahmed, and S. Farhana (2008), "Walking Hexapod Robot in Disaster Recovery: Developing Algorithm for Terrain Negotiation and Navigation".*
- G.Evangelista (2014), "Design and Modeling of a Mobile Research Platform based on Hexapod Robot with Embedded System and Interactive Control".*
- R. Chung, Y. Zhang and S. Chen (2015), "Full pipelined CORDIC-based inverse kinematic FPGA design for biped robots".*
- Z.Yili, S. Hanxu, J-Qingxuan and S. Guozhen (2008), "Kinematics Control for a 6-DOF Space Manipulator Based on ARM Processor and FPGA Co-processor".*
- D.Henrich and T. Höniger (1997), "Parallel Processing Approaches In Robotics".*
- B.Naji, K.Abbes, C.Abdelmoula and M.Masmoudi (2016), "Successfully Designing FPGA-Card for Mobile Robot Research".*
- F.Tedeschi and G. Carbone (2014), "Design Issues for Hexapod Walking Robots".*
- A.Roennau, G.Heppner, M.Nowicki and R.Dillman (2014), "LAURON V: A Versatile Six-Legged Walking Robot with Advanced Maneuverability".*
- M. Arora, S. Chauhan and L. Bagga (2012), "FPGA Prototyping of Hardware Implementation of CORDIC Algorithm".*
- P. Meher, J.Valls, T. Juang, k. Sridharan and k. Maharatna (2009). "50 Years of CORDIC: Algorithms, Architectures and Applications".*
- X.Hu, R. Harber and S. Bass (1991), "Expanding the Range of Convergence of the CORDIC Algorithm".*

## ENLACES WEB

Matlab. (2017). El lenguaje del cálculo técnico. Recuperado de <https://es.mathworks.com/products/matlab.html>

Xilinx (2017). ISE Simulator (ISIM). Recuperado de <https://www.xilinx.com/products/design-tools/isim.html>

University of Wisconsin-Madison (2015). Computer Sciences, Floating Point Arithmetic. Recuperado de <http://pages.cs.wisc.edu/~smoler/x86text/lect.notes/arith.flpt.html>.

# **ANEXOS**

[illegible]



## Anexo 02 Arquitectura principal e instanciación de componentes

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.cordic2_pkg.all;

entity SM_1 is
    port (
        clk      : in    std_logic;
        rst      : in    std_logic;
        px        : in    std_logic_vector(31 downto 0);
        py        : in    std_logic_vector(31 downto 0);
        pz        : in    std_logic_vector(31 downto 0);
        en        : in    std_logic;
        q1        : out   std_logic_vector(31 downto 0);
        q1_r      : out   std_logic;
        q2        : out   std_logic_vector(31 downto 0);
        q2_r      : out   std_logic;
        q3        : out   std_logic_vector(31 downto 0);
        q3_r      : out   std_logic
    );
end SM_1;

architecture Behavioral of SM_1 is
    -- In/Out
    signal s_rst      : std_logic := '0';
    signal s_px       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_py       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_pz       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_q1       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_q1r      : std_logic := '0';
    signal s_q3       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_q3r      : std_logic := '0';
    signal s_q2       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_q2r      : std_logic := '0';
    -- state Machine
    type mach1_t is (e_idle, e_c1, e_c2, e_c3, e_c4);
    signal st_m1      : mach1_t := e_idle;
    signal sn_m1      : mach1_t := e_idle;
    -- signaling
    signal w_reg      : std_logic_vector(31 downto 0) := (others => '0');
    signal w_rdy      : std_logic := '0';
    signal r_reg      : std_logic_vector(31 downto 0) := (others => '0');
    signal r_rdy      : std_logic := '0';
    signal d_reg      : std_logic_vector(31 downto 0) := (others => '0');
    signal d_rdy      : std_logic := '0';
    signal r_m_l1     : std_logic_vector(31 downto 0) := (others => '0');
    signal root1      : std_logic_vector(31 downto 0) := (others => '0');
    signal root2      : std_logic_vector(31 downto 0) := (others => '0');
    -- cordic CC1
```

```

signal c1_en      : std_logic := '0';
signal c1_x0      : std_logic_vector(31 downto 0) := (others => '0');
signal c1_y0      : std_logic_vector(31 downto 0) := (others => '0');
signal c1_z0      : std_logic_vector(31 downto 0) := (others => '0');
signal c1_mo      : std_logic := '0';
signal c1_dr      : std_logic := '0';
signal c1_xn      : std_logic_vector(31 downto 0) := (others => '0');
--signal c1_yn    : std_logic_vector(31 downto 0) := (others => '0');
signal c1_zn      : std_logic_vector(31 downto 0) := (others => '0');
-- cordic CH2
signal c2_en      : std_logic := '0';
signal c2_rkh     : std_logic_vector(31 downto 0) := (others => '0');
signal c2_dr      : std_logic := '0';
-- cordic CC3
signal c3_en      : std_logic := '0';
signal c3_x0      : std_logic_vector(31 downto 0) := (others => '0');
signal c3_y0      : std_logic_vector(31 downto 0) := (others => '0');
signal c3_mo      : std_logic := '0';
signal c3_dr      : std_logic := '0';
--signal c3_xn    : std_logic_vector(31 downto 0) := (others => '0');
--signal c3_yn    : std_logic_vector(31 downto 0) := (others => '0');
signal c3_zn      : std_logic_vector(31 downto 0) := (others => '0');
-- cordic CC4
signal c4_dr      : std_logic := '0';
signal c4_xn      : std_logic_vector(31 downto 0) := (others => '0');
-- cordic CC5
signal c5_dr      : std_logic := '0';
signal c5_xn      : std_logic_vector(31 downto 0) := (others => '0');
signal c5_yn      : std_logic_vector(31 downto 0) := (others => '0');
-- cordic CC7
signal c6_dr      : std_logic := '0';
signal c6_zn      : std_logic_vector(31 downto 0) := (others => '0');
-- cordic CC7
signal c7_dr      : std_logic := '0';
signal c7_zn      : std_logic_vector(31 downto 0) := (others => '0');
-- cordic CC8
signal c8_dr      : std_logic := '0';
signal c8_xn      : std_logic_vector(31 downto 0) := (others => '0');
-- cordic CC9
signal c9_dr      : std_logic := '0';
signal c9_zn      : std_logic_vector(31 downto 0) := (others => '0');
-- mult + adder 1
signal mu1_x      : std_logic_vector(31 downto 0) := (others => '0');
signal mu1_y      : std_logic_vector(31 downto 0) := (others => '0');
signal mu1_z      : std_logic_vector(31 downto 0) := (others => '0');
signal ad1_r      : std_logic_vector(31 downto 0) := (others => '0');
signal ad1_x      : std_logic_vector(31 downto 0) := (others => '0');
signal ad1_y      : std_logic_vector(31 downto 0) := (others => '0');
signal ad1_s      : std_logic := '0';
signal ad1_z      : std_logic_vector(31 downto 0) := (others => '0');
-- mult 2
signal mu2_x      : std_logic_vector(31 downto 0) := (others => '0');
signal mu2_z      : std_logic_vector(31 downto 0) := (others => '0');
-- adder 3

```

```
signal cos_p_f : std_logic_vector(31 downto 0) := (others => '0');
```

```
begin
```

```
--
```

```
s_rst <= rst;
```

```
s_px <= px;
```

```
s_py <= py;
```

```
s_pz <= pz;
```

```
q1 <= c1_zn;
```

```
q1_r <= c1_dr;
```

```
q3 <= s_q3;
```

```
q3_r <= s_q3r;
```

```
q2 <= s_q2;
```

```
q2_r <= s_q2r;
```

```
-- Mult + adder 1
```

```
ad1_y <= ad1_r;
```

```
ad1_x <= mul_z;
```

```
-- cordic circular 1
```

```
c1_en <= en;
```

```
c1_mo <= '1';
```

```
c1_x0 <= s_px;
```

```
c1_y0 <= s_py;
```

```
c1_z0 <= FP_val0;
```

```
-- Mapping
```

```
CC1: CORDIC2_1 port map (
```

```
    clk      => clk,
```

```
    rst      => s_rst,
```

```
    en       => c1_en,
```

```
    x0       => c1_x0,
```

```
    y0       => c1_y0,
```

```
    z0       => c1_z0,
```

```
    mode     => c1_mo,
```

```
    data_ready => c1_dr,
```

```
    xn       => c1_xn,
```

```
    yn       => open,
```

```
    zn       => c1_zn
```

```
);
```

```
--
```

```
Mult_1: FP_MULT port map (
```

```
    Xnum => mul_x,
```

```
    Ynum =>    mul_y,
```

```
    Mult =>    mul_z
```

```
);
```

```
Add_1: FP_add port map (
```

```
    Xnum => ad1_x,
```

```
    Ynum =>    ad1_y,
```

```
    SubnA =>    ad1_s,
```

```
    Zadd  =>    ad1_z
```

```
);
```

```
--
```

```

Add_2: FP_add port map (
    Xnum => r_reg,
    Ynum =>    FP_valL1,
    SubnA =>    '1',
    Zadd  =>    r_m_l1
);

CC4: CORDIC2_1 port map (
    clk      => clk,
    rst      => s_rst,
    en       => r_rdy,
    x0       => r_m_l1,
    y0       => s_pz,
    z0       => FP_val0,
    mode     => '1',
    data_ready => c4_dr,
    xn       => c4_xn, -- BK
    yn       => open,
    zn       => open
);

Mult_2: FP_MULT port map (
    Xnum => c4_xn,
    Ynum =>    val_iK,
    Mult  =>    mu2_z -- B
);

CC7: CORDIC2_0 port map (
    clk      => clk,
    rst      => s_rst,
    en       => c4_dr,
    x0       => mu2_z,
    y0       => s_pz,
    z0       => FP_val0,
    mode     => '1',
    data_ready => c7_dr,
    xn       => open,
    yn       => open,
    zn       => c7_zn
);

CC5: CORDIC2_1 port map (
    clk      => clk,
    rst      => s_rst,
    en       => s_q3r,
    x0       => val_iK,
    y0       => FP_val0,
    z0       => s_q3,
    mode     => '0',
    data_ready => c5_dr,
    xn       => c5_xn, --
    yn       => c5_yn,
    zn       => open
);

```

```

Add_3: FP_add port map (
    Xnum => d_reg,
    Ynum =>    FP_valF,
    SubnA =>    '0',
    Zadd  =>    cos_p_f
);

CC6: CORDIC2_1 port map (
    clk      => clk,
    rst      => s_rst,
    en       => c5_dr,
    x0       => cos_p_f,
    y0       => c5_yn,
    z0       => FP_val0,
    mode     => '1',
    data_ready => c6_dr,
    xn       => open, --
    yn       => open,
    zn       => c6_zn
);

-- Calculo Q3
c2_en <= d_rdy;
w_rdy <= d_rdy;
w_reg <= d_reg;

CH2: CORDIC2_2 port map (
    clk      => clk,
    rst      => s_rst,
    en       => c2_en,
    x0       => val_1z,
    y0       => w_reg,
    data_ready => c2_dr,
    xn       => c2_rkh
);

Mult_3: FP_MULT port map (
    Xnum => c2_rkh,
    Ynum =>    val_iKh,
    Mult =>    root1
);

CC3: CORDIC2_1 port map (
    clk      => clk,
    rst      => s_rst,
    en       => c2_dr,
    x0       => c3_x0,
    y0       => c3_y0,
    z0       => FP_val0,
    mode     => '1',
    data_ready => s_q3r,
    xn       => open,
    yn       => open,

```

```

        zn            => s_q3
    );

    c3_y0(30 downto 0) <= root1(30 downto 0);
    c3_x0 <= '0' & w_reg(30 downto 0);

    CC3_in: process (w_reg, root1)
    begin
        if w_reg(31) = '1' then
            c3_y0(31) <= not root1(31);
        else
            c3_y0(31) <= root1(31);
        end if;
    end process CC3_in;

    -- Calculo Q2
    CH8: CORDIC2_2 port map (
        clk      => clk,
        rst      => s_rst,
        en       => c7_dr,
        x0       => val_1z,
        y0       => c7_zn,
        data_ready => c8_dr,
        xn       => c8_xn
    );

    Mult_4: FP_MULT port map (
        Xnum => c8_xn,
        Ynum => val_iKh,
        Mult => root2
    );

    CC9: CORDIC2_1 port map (
        clk      => clk,
        rst      => s_rst,
        en       => c8_dr,
        x0       => root2,
        y0       => c7_zn,
        z0       => FP_val0,
        mode     => '1',
        data_ready => c9_dr,
        xn       => open,
        yn       => open,
        zn       => c9_zn
    );

    Add_4: FP_add port map (
        Xnum => c9_zn,
        Ynum => c6_zn,
        SubnA => '1',
        Zadd  => s_q2
    );

    s_q2r <= c6_dr;

```

--

```
Adder_reg: process (s_rst, clk, st_m1)
begin
    if s_rst = '1' then
        ad1_r <= FP_valc3;
    elsif rising_edge(clk) then
        if st_m1 = e_idle then
            ad1_r <= FP_valc3;
        else
            ad1_r <= ad1_z;
        end if;
    end if;
end process Adder_reg;
```

```
State_Machine1: process (s_rst, clk)
begin
    if s_rst = '1' then
        mul_x <= (others => '0');
        mul_y <= val_iK;
    elsif falling_edge(clk) then
        case (st_m1) is
            when e_idle =>
                d_rdy <= '0';
                mul_x <= c1_xn;
                mul_y <= val_iK;
                if c1_dr = '1' then
                    sn_m1 <= e_c1;
                end if;
            when e_c1 =>
                r_reg <= mul_z;
                r_rdy <= '1';
                mul_x <= s_pz;
                mul_y <= s_pz;
                sn_m1 <= e_c2;
            when e_c2 =>
                r_rdy <= '0';
                mul_x <= r_reg;
                mul_y <= r_reg;
                sn_m1 <= e_c3;
            when e_c3 =>
                mul_x <= r_reg;
                mul_y <= FP_valc2;
                sn_m1 <= e_c4;
            when e_c4 =>
                mul_x <= ad1_r;
                mul_y <= FP_valc9;
                d_reg <= mul_z;
                d_rdy <= '1';
                sn_m1 <= e_idle;
            end case;
        end if;
    end process State_Machine1;
```

```
state_control1: process (clk, s_rst, sn_m1)
begin
    if s_rst = '1' then
        st_m1 <= e_idle;
    elsif rising_edge(clk) then
        st_m1 <= sn_m1;
    end if;
end process state_control1;
end Behavioral;
```



### Anexo 03: Entidad CORDIC 2.0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.cordic2_pkg.ALL;

entity CORDIC2_0 is
    port (
        clk      : in    std_logic;
        rst      : in    std_logic;
        en       : in    std_logic;
        x0       : in    std_logic_vector(31 downto 0);
        y0       : in    std_logic_vector(31 downto 0);
        z0       : in    std_logic_vector(31 downto 0);
        mode     : in    std_logic;      -- 0 : Circ, 1 : Vect
        data_ready : out   std_logic;
        xn       : out   std_logic_vector(31 downto 0);
        yn       : out   std_logic_vector(31 downto 0);
        zn       : out   std_logic_vector(31 downto 0);
    );
end CORDIC2_0;

architecture Behavioral of CORDIC2_0 is
    --
    signal s_rst      : std_logic := '0';
    signal s_en       : std_logic := '0';
    signal s_ready    : std_logic := '0';
    signal iter       : integer range 0 to c_niter - 1 := 0;
    type smach_t is (e_idle, e_in, e_op, e_out);
    signal smach      : smach_t := e_idle;
    signal snext      : smach_t := e_idle;
    signal x_reg      : std_logic_vector(31 downto 0) := (others => '0');
    signal y_reg      : std_logic_vector(31 downto 0) := (others => '0');
    signal z_reg      : std_logic_vector(31 downto 0) := (others => '0');
    signal sum_reg    : std_logic_vector(31 downto 0) := (others => '0');
    -- In/Out Signaling
    signal s_x0       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_y0       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_z0       : std_logic_vector(31 downto 0) := (others => '0');
    signal d_sig      : std_logic := '0';
    signal d_sign     : std_logic := '0';
    signal s_xn       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_yn       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_zn       : std_logic_vector(31 downto 0) := (others => '0');
    --signal s_xi      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_yi       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_zi       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_xir      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_yir      : std_logic_vector(31 downto 0) := (others => '0');
    begin
        --
        s_rst <= rst;
        s_x0 <= x0;
```

```

s_y0 <= y0;
s_z0 <= z0;
xn <= s_xn;
yn <= s_yn;
zn <= s_zn;
s_en <= en;
data_ready <= s_ready;
--

Mode_sel: process (mode, z_reg, y_reg)
begin
    if mode = '0' then
        d_sig <= z_reg(31);
        d_sign <= not z_reg(31);
    else
        d_sig <= not y_reg(31);
        d_sign <= y_reg(31);
    end if;
end process Mode_sel;

state_machine: process (clk, s_rst, smach)
begin
    if s_rst = '1' then
        snext <= e_idle;
        x_reg <= (others => '0');
        y_reg <= (others => '0');
        z_reg <= (others => '0');
        sum_reg <= atan2(0);
        iter <= 0;
        s_ready <= '0';
    elsif rising_edge(clk) then
        case smach is
            when e_idle =>
                s_ready <= '0';
                if s_en = '1' then
                    snext <= e_in;
                else
                    snext <= e_idle;
                end if;
            when e_in =>
                x_reg <= s_x0;
                y_reg <= s_y0;
                z_reg <= s_z0;
                sum_reg <= val_1z;
                snext <= e_op;
                s_xir <= s_x0;
                s_yir <= s_y0;
                iter <= iter + 1;
            when e_op =>
                s_xir(31) <= x_reg(31);
                s_xir(30 downto 23) <=
std_logic_vector(unsigned(x_reg(30 downto 23)) - iter);
                s_xir(22 downto 0) <= x_reg(22 downto 0);
                sum_reg(31) <= val_1z(31);

```

```

                                sum_reg(30 downto 23) <=
std_logic_vector(unsigned(val_1z(30 downto 23)) - iter);
                                sum_reg(22 downto 0) <= val_1z(22 downto 0);
                                y_reg <= s_yi;
                                z_reg <= s_zi;
                                if iter < c_niter - 1 then
                                    iter    <= iter + 1;
                                    snext   <= e_op;
                                else
                                    snext <= e_out;
                                end if;
                                when e_out =>
                                    iter <= 0;
                                    snext <= e_idle;
                                    s_ready <= '1';
                                end case;
                            end if;
end process state_machine;

state_next: process (clk, s_rst, smach)
begin
    if s_rst = '1' then
        smach <= e_idle;
    elsif falling_edge(clk) then
        smach <= snext;
    end if;
end process state_next;

-- Sumadores
Sum_Y: FP_add port map (
    Xnum  => y_reg,
    Ynum  => s_xir,
    SubnA => d_sig,
    Zadd  => s_yi
);

Sum_Z: FP_add port map (
    Xnum  => z_reg,
    Ynum  => sum_reg,
    SubnA => d_sign,
    Zadd  => s_zi
);

Out_port: process (clk, s_rst, smach)
begin
    if s_rst = '1' then
        s_xn <= (others => '0');
        s_yn <= (others => '0');
        s_zn <= (others => '0');
    elsif falling_edge(clk) and snext = e_out then
        s_xn <= x_reg;
        s_yn <= s_yi;
        s_zn <= s_zi;
    end if;
end process;

```

```
end process Out_port;  
end Behavioral;
```

## Anexo 04: Entidad CORDIC 2.1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.cordic2_pkg.ALL;

entity CORDIC2_1 is
    port (
        clk      : in    std_logic;
        rst      : in    std_logic;
        en       : in    std_logic;
        x0       : in    std_logic_vector(31 downto 0);
        y0       : in    std_logic_vector(31 downto 0);
        z0       : in    std_logic_vector(31 downto 0);
        mode     : in    std_logic;      -- 0 : Circ, 1 : Vect
        data_ready : out   std_logic;
        xn       : out   std_logic_vector(31 downto 0);
        yn       : out   std_logic_vector(31 downto 0);
        zn       : out   std_logic_vector(31 downto 0);
    );
end CORDIC2_1;

architecture Behavioral of CORDIC2_1 is
    --
    signal s_en      : std_logic := '0';
    signal s_rst     : std_logic := '0';
    signal s_ready   : std_logic := '0';
    signal iter      : integer range 0 to c_niter - 1 := 0;
    type smach_t is (e_idle, e_in, e_op, e_out);
    signal smach     : smach_t := e_idle;
    signal snext     : smach_t := e_idle;
    signal x_reg     : std_logic_vector(31 downto 0) := (others => '0');
    signal y_reg     : std_logic_vector(31 downto 0) := (others => '0');
    signal z_reg     : std_logic_vector(31 downto 0) := (others => '0');
    signal sum_reg   : std_logic_vector(31 downto 0) := (others => '0');
    -- In/Out Signaling
    signal s_x0      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_y0      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_z0      : std_logic_vector(31 downto 0) := (others => '0');
    signal d_sig     : std_logic := '0';
    signal d_sign    : std_logic := '0';
    signal s_xn      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_yn      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_zn      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_xi      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_yi      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_zi      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_xir     : std_logic_vector(31 downto 0) := (others => '0');
    signal s_yir     : std_logic_vector(31 downto 0) := (others => '0');
    begin

    --
    s_rst <= rst;
```

```

s_x0 <= x0;
s_y0 <= y0;
s_z0 <= z0;
xn <= s_xn;
yn <= s_yn;
zn <= s_zn;
s_en <= en;
data_ready <= s_ready;
--

Mode_sel: process (mode, z_reg, y_reg)
begin
    if mode = '0' then
        d_sig <= z_reg(31);
        d_sign <= not z_reg(31);
    else
        d_sig <= not y_reg(31);
        d_sign <= y_reg(31);
    end if;
end process Mode_sel;

state_machine: process (clk, s_rst, smach)
begin
    if s_rst = '1' then
        snext <= e_idle;
        x_reg <= (others => '0');
        y_reg <= (others => '0');
        z_reg <= (others => '0');
        sum_reg <= atan2(0);
        iter <= 0;
        s_ready <= '0';
    elsif rising_edge(clk) then
        case smach is
            when e_idle =>
                s_ready <= '0';
                if s_en = '1' then
                    snext <= e_in;
                else
                    snext <= e_idle;
                end if;
            when e_in =>
                x_reg <= s_x0;
                y_reg <= s_y0;
                z_reg <= s_z0;
                sum_reg <= atan2(0);
                snext <= e_op;
                s_xir <= s_x0;
                s_yir <= s_y0;
                iter <= iter + 1;
            when e_op =>
                s_xir(31) <= s_xi(31);
                s_yir(31) <= s_yi(31);
                s_xir(30 downto 23) <=
std_logic_vector(unsigned(s_xi(30 downto 23)) - iter);

```

```

                                s_yir(30 downto 23) <=
std_logic_vector(unsigned(s_yi(30 downto 23)) - iter);
                                s_xir(22 downto 0) <= s_xi(22 downto 0);
                                s_yir(22 downto 0) <= s_yi(22 downto 0);
                                x_reg <= s_xi;
                                y_reg <= s_yi;
                                z_reg <= s_zi;
                                sum_reg <= atan2(iter);
                                if iter < c_niter - 1 then
                                    iter <= iter + 1;
                                    snext <= e_op;
                                else
                                    snext <= e_out;
                                end if;
                                when e_out =>
                                    iter <= 0;
                                    snext <= e_idle;
                                    s_ready <= '1';
                                end case;
                            end if;
                        end process state_machine;

state_next: process (clk, s_rst, smach)
begin
    if s_rst = '1' then
        smach <= e_idle;
    elsif falling_edge(clk) then
        smach <= snext;
    end if;
end process state_next;

-- Sumadores
Sum_X: FP_add port map (
    Xnum => x_reg,
    Ynum => s_yir,
    SubnA => d_sign,
    Zadd => s_xi
);

Sum_Y: FP_add port map (
    Xnum => y_reg,
    Ynum => s_xir,
    SubnA => d_sig,
    Zadd => s_yi
);

Sum_Z: FP_add port map (
    Xnum => z_reg,
    Ynum => sum_reg,
    SubnA => d_sign,
    Zadd => s_zi
);

Out_port: process (clk, s_rst, smach)

```

```

begin
  if s_rst = '1' then
    s_xn <= (others => '0');
    s_yn <= (others => '0');
    s_zn <= (others => '0');
  elsif falling_edge(clk) and snext = e_out then
    s_xn <= s_xi;
    s_yn <= s_yi;
    s_zn <= s_zi;
  end if;
end process Out_port;
end Behavioral;

```



## Anexo 05: Entidad CORDIC 2.2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.cordic2_pkg.ALL;

entity CORDIC2_2 is
    port (
        clk      : in    std_logic;
        rst      : in    std_logic;
        en       : in    std_logic;
        x0       : in    std_logic_vector(31 downto 0);
        y0       : in    std_logic_vector(31 downto 0);
        data_ready : out   std_logic;
        xn       : out   std_logic_vector(31 downto 0);
        --yn     : out   std_logic_vector(31 downto 0);
    );
end CORDIC2_2;

architecture Behavioral of CORDIC2_2 is
    --
    signal s_rst      : std_logic := '0';
    signal s_en       : std_logic := '0';
    signal s_ready    : std_logic := '0';
    signal iter       : integer range 0 to c_hipiter - 1 := 0;
    type smach_t is (e_idle, e_in, e_op, e_out);
    signal smach      : smach_t := e_idle;
    signal snext      : smach_t := e_idle;
    signal x_reg      : std_logic_vector(31 downto 0) := (others => '0');
    signal y_reg      : std_logic_vector(31 downto 0) := (others => '0');
    -- In/Out Signaling
    signal s_x0       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_y0       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_xn       : std_logic_vector(31 downto 0) := (others => '0');
    --signal s_yn     : std_logic_vector(31 downto 0) := (others => '0');
    signal s_xi       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_yi       : std_logic_vector(31 downto 0) := (others => '0');
    signal s_xi1      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_yi1      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_xir      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_yir      : std_logic_vector(31 downto 0) := (others => '0');
    signal s_sig      : std_logic := '0';
    signal d_nsig     : std_logic := '0';
    begin
        --
        s_rst <= rst;
        s_x0 <= x0;
        s_y0 <= y0;
        xn <= s_xn;
        --yn <= s_yn;
        s_en <= en;
        data_ready <= s_ready;
        d_nsig <= not y_reg(31);
    end
end
```

```

--
Ssign: process (y_reg,iter)
begin
    if iter <= c_M + 1 then
        s_sig <= y_reg(31);
    else
        s_sig <= not y_reg(31);
    end if;
end process Ssign;

state_machine: process (clk, s_rst, smach)
begin
    if s_rst = '1' then
        snext <= e_idle;
        x_reg <= (others => '0');
        y_reg <= (others => '0');
        iter <= 0;
        s_ready <= '0';
    elsif rising_edge(clk) then
        case smach is
            when e_idle =>
                s_ready <= '0';
                iter <= 0;
                if s_en = '1' then
                    snext <= e_in;
                else
                    snext <= e_idle;
                end if;
            when e_in =>
                x_reg <= s_x0;
                y_reg <= s_y0;
                s_xir(31) <= s_x0(31);
                s_yir(31) <= s_y0(31);
                s_xir(30 downto 23) <=
std_logic_vector(unsigned(s_x0(30 downto 23)) - 6 + iter);
                s_yir(30 downto 23) <=
std_logic_vector(unsigned(s_y0(30 downto 23)) - 6 + iter);
                s_xir(22 downto 0) <= s_x0(22 downto 0);
                s_yir(22 downto 0) <= s_y0(22 downto 0);
                snext <= e_op;
                iter <= iter + 1;
            when e_op =>
                if iter <= c_M + 1 then
                    x_reg <= s_xi1;
                    y_reg <= s_yi1;
                    s_xir(31) <= s_xi1(31);
                    s_yir(31) <= s_yi1(31);
                    s_xir(30 downto 23) <=
std_logic_vector(unsigned(s_xi1(30 downto 23)) - 6 + iter);
                    s_yir(30 downto 23) <=
std_logic_vector(unsigned(s_yi1(30 downto 23)) - 6 + iter);
                    s_xir(22 downto 0) <= s_xi1(22 downto 0);
                    s_yir(22 downto 0) <= s_yi1(22 downto 0);
                else

```

```

        x_reg <= s_xi;
        y_reg <= s_yi;
        s_xir(31) <= s_xi(31);
        s_yir(31) <= s_yi(31);
        s_xir(30 downto 23) <=
std_logic_vector(unsigned(s_xi(30 downto 23)) - iter + 4);
        s_yir(30 downto 23) <=
std_logic_vector(unsigned(s_yi(30 downto 23)) - iter + 4);
        s_xir(22 downto 0) <= s_xi(22 downto 0);
        s_yir(22 downto 0) <= s_yi(22 downto 0);
    end if;
    if iter < c_hipiter - 1 then
        iter    <= iter + 1;
        snext   <= e_op;
    else
        snext <= e_out;
    end if;
    when e_out =>
        iter <= 0;
        snext <= e_idle;
        s_ready <= '1';
    end case;
end if;
end process state_machine;

state_next: process (clk, s_rst, smach)
begin
    if s_rst = '1' then
        smach <= e_idle;
    elsif falling_edge(clk) then
        smach <= snext;
    end if;
end process state_next;

-- Sumadores
Sum_X: FP_add port map (
    Xnum  => x_reg,
    Ynum  => s_yir,
    SubnA => s_sig,
    Zadd  => s_xi
);

Sum_Y: FP_add port map (
    Xnum  => y_reg,
    Ynum  => s_xir,
    SubnA => s_sig,
    Zadd  => s_yi
);

Sum_X1: FP_add port map (
    Xnum  => s_xi,
    Ynum  => y_reg,
    SubnA => d_nsig,
    Zadd  => s_xi1

```

```

);

Sum_Y1: FP_add port map (
    Xnum => s_yi,
    Ynum => x_reg,
    SubnA => d_nsig,
    Zadd  => s_yi1
);

Out_port: process (clk, s_rst, smach)
begin
    if s_rst = '1' then
        s_xn <= (others => '0');
        --s_yn <= (others => '0');
    elsif falling_edge(clk) and snext = e_out then
        s_xn <= s_xi;
        --s_yn <= s_yi;
    end if;
end process Out_port;
end Behavioral;

```

## Anexo 06: Paquete de constantes y componentes

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package cordic2_pkg is

-- Look-up Table
constant c_niter : integer := 16;
constant c_hipiter : integer range 0 to 255 := 22;
constant c_M : integer range 0 to 255 := 4;
type atan2_t is array (0 to c_niter - 1) of std_logic_vector(31 downto 0);
constant atan2 : atan2_t := (X"3f490fdb", X"3eed6338", X"3e7adbb0", X"3dfeadd5",
X"3d7faade", X"3cffeaae", X"3c7ffaab", X"3bffeab", X"3b7fffaa", X"3affffef", X"3a7ffffb",
X"39ffffff", X"39800000", X"39000000", X"38800000", X"37800000");
constant val_1z : std_logic_vector(31 downto 0) := X"3F800000";
constant val_iK : std_logic_vector(31 downto 0) := X"3f1b74ed";
constant val_iKh : std_logic_vector(31 downto 0) := X"43779f21"; -- ikh = 47.6216
constant FP_val0 : std_logic_vector(31 downto 0) := X"00000000";
constant FP_valL1 : std_logic_vector(31 downto 0) := X"3ce147ae";
constant FP_valc3 : std_logic_vector(31 downto 0) := X"bca04387"; -- C3 = 11^2 - 12^2 - 13^3
constant FP_valc2 : std_logic_vector(31 downto 0) := X"bd6147ae"; -- C2 = - 2L1
constant FP_valc8 : std_logic_vector(31 downto 0) := X"3ca5d319"; -- C8 = 2L2L3
constant FP_valc9 : std_logic_vector(31 downto 0) := X"42459b3d"; -- C9 = 1 / C8
constant FP_valF : std_logic_vector(31 downto 0) := X"3f6a90af";

component CORDIC2_2
port (
    clk : in std_logic;
    rst : in std_logic;
    en : in std_logic;
    x0 : in std_logic_vector(31 downto 0);
    y0 : in std_logic_vector(31 downto 0);
    data_ready : out std_logic;
    xn : out std_logic_vector(31 downto 0);
    --yn : out std_logic_vector(31 downto 0);
);

end component CORDIC2_2;

component CORDIC2_1
port (
    clk : in std_logic;
    rst : in std_logic;
    en : in std_logic;
    x0 : in std_logic_vector(31 downto 0);
    y0 : in std_logic_vector(31 downto 0);
    z0 : in std_logic_vector(31 downto 0);
    mode : in std_logic; -- 0 : Circ, 1 : Vect
    data_ready : out std_logic;
    xn : out std_logic_vector(31 downto 0);
    yn : out std_logic_vector(31 downto 0);
    zn : out std_logic_vector(31 downto 0);
);
```

```

end component CORDIC2_1;

component CORDIC2_0
    port (
        clk      : in    std_logic;
        rst      : in    std_logic;
        en       : in    std_logic;
        x0       : in    std_logic_vector(31 downto 0);
        y0       : in    std_logic_vector(31 downto 0);
        z0       : in    std_logic_vector(31 downto 0);
        mode     : in    std_logic;      -- 0 : Circ, 1 : Vect
        data_ready : out   std_logic;
        xn       : out   std_logic_vector(31 downto 0);
        yn       : out   std_logic_vector(31 downto 0);
        zn       : out   std_logic_vector(31 downto 0)
    );
end component CORDIC2_0;

component FP_MULT
    port (
        Xnum : in    std_logic_vector(31 downto 0) := (others => '0');
        Ynum : in    std_logic_vector(31 downto 0) := (others => '0');
        Mult : out   std_logic_vector(31 downto 0)
    );
end component FP_MULT;

component FP_add
    port (
        Xnum : in    std_logic_vector(31 downto 0);
        Ynum : in    std_logic_vector(31 downto 0);
        SubnA : in    std_logic;
        Zadd  : out   std_logic_vector(31 downto 0)
    );
end component FP_add;

component FP_add2
    port (
        Xnum : in    std_logic_vector(31 downto 0);
        Ynum : in    std_logic_vector(31 downto 0);
        Zadd : out   std_logic_vector(31 downto 0)
    );
end component FP_add2;

component FP_Gt
    port (
        Xnum : in    std_logic_vector(31 downto 0);
        Ynum : in    std_logic_vector(31 downto 0);
        XgtY : out   std_logic
    );
end component FP_Gt;

end cordic2_pkg;

```

## Anexo 07: TESTBENCH

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;
ENTITY SM_tb IS
END SM_tb;

ARCHITECTURE behavior OF SM_tb IS
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT SM_1
    PORT(
        clk : IN  std_logic;
        rst : IN  std_logic;
        px : IN  std_logic_vector(31 downto 0);
        py : IN  std_logic_vector(31 downto 0);
        pz : IN  std_logic_vector(31 downto 0);
        en : IN  std_logic;
        q1 : OUT std_logic_vector(31 downto 0);
        q1_r : OUT std_logic;
        q2 : OUT std_logic_vector(31 downto 0);
        q2_r : OUT std_logic;
        q3 : OUT std_logic_vector(31 downto 0);
        q3_r : OUT std_logic
    );
END COMPONENT;

    --Inputs
    signal clk : std_logic := '0';
    signal rst : std_logic := '0';
    signal px : std_logic_vector(31 downto 0) := (others => '0');
    signal py : std_logic_vector(31 downto 0) := (others => '0');
    signal pz : std_logic_vector(31 downto 0) := (others => '0');
    signal en : std_logic := '0';
```

```

        --Outputs
signal q1 : std_logic_vector(31 downto 0);
signal q1_r : std_logic;
signal q2 : std_logic_vector(31 downto 0);
signal q2_r : std_logic;
signal q3 : std_logic_vector(31 downto 0);
signal q3_r : std_logic;

-- Clock period definitions
constant clk_period : time := 20 ns;
BEGIN

    -- Instantiate the Unit Under Test (UUT)
uut: SM_1 PORT MAP (
    clk => clk,
    rst => rst,
    px => px,
    py => py,
    pz => pz,
    en => en,
    q1 => q1,
    q1_r => q1_r,
    q2 => q2,
    q2_r => q2_r,
    q3 => q3,
    q3_r => q3_r
);

-- Clock process definitions
clk_process :process
begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
end process;
--stimulus_process :process

```



```

-- -- Stimulus process
-- begin
--   -- hold reset state for 100 ns.
--   wait for 5 ns;
--   -- insert stimulus here
--
--   wait for clk_period*10;
--   wait;
-- end process;
barrido : process
begin
--LINEA 1
    px <= x"3DFD8ADB";
    py <= x"3D23D70A";
    pz <= x"BDD73EAB";
    en <= not en;
        wait for 20 ns;
            en <= not en;
                wait for clk_period*105;
--LINEA 2
    px <= x"3DFEB7CC";
    py <= x"3D207C0C";
    pz <= x"BDD73EAB";
        en <= not en;
            wait for 20 ns;
                en <= not en;
wait for clk_period*105;
--LINEA 3
    px <= x"3DFFE436";
    py <= x"3D1D2128";
    pz <= x"BDD73EAB";
        en <= not en;
            wait for 20 ns;
                en <= not en;
wait for clk_period*105;
--LINEA 4

```

```

        px <= x"3E008894";
    py <= x"3D19C62A";
    pz <= x"BDD73EAB";

    en <= not en;

    wait for 20 ns;

    en <= not en;
wait for clk_period*105;
--LINEA 5

        px <= x"3E011EC9";
    py <= x"3D166B47";
    pz <= x"BDD73EAB";

    en <= not en;

    wait for 20 ns;

    en <= not en;
wait for clk_period*105;
--LINEA 6

        px <= x"3E01B542";
    py <= x"3D131048";
    pz <= x"BDD73EAB";

    en <= not en;

    wait for 20 ns;

    en <= not en;
wait for clk_period*105;
--LINEA 7

        px <= x"3E024B77";
    py <= x"3D0FB565";
    pz <= x"BDD73EAB";

    en <= not en;

    wait for 20 ns;

    en <= not en;
wait for clk_period*105;
--LINEA 8

        px <= x"3E02E1EF";
    py <= x"3D0C5A66";
    pz <= x"BDD73EAB";

    en <= not en;

```

```

        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 9
        px <= x"3E037868";
        py <= x"3D08FF83";
        pz <= x"BDD73EAB";
        en <= not en;
        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 10
        px <= x"3E040E9D";
        py <= x"3D05A485";
        pz <= x"BDD73EAB";
        en <= not en;
        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 11
        px <= x"3E04A516";
        py <= x"3D0249A1";
        pz <= x"BDD73EAB";
        en <= not en;
        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 12
        px <= x"3E01B542";
        py <= x"3D131048";
        pz <= x"BDB67A10";
        en <= not en;
        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 13

```

```

        px <= x"3DFD8ADB";
    py <= x"3D23D70A";
    pz <= x"BDA617C2";

    en <= not en;
    wait for 20 ns;
    en <= not en;
wait for clk_period*105;
--LINEA 14
        px <= x"3DF7AB32";
    py <= x"3D349DCC";
    pz <= x"BDA617C2";

    en <= not en;
    wait for 20 ns;
    en <= not en;
wait for clk_period*105;
--LINEA 15
        px <= x"3DF1CB8A";
    py <= x"3D456473";
    pz <= x"BDB67A10";

    en <= not en;
    wait for 20 ns;
    en <= not en;
wait for clk_period*105;
--LINEA 16
        px <= x"3DEBEBE1";
    py <= x"3D562B35";
    pz <= x"BDD73EAB";

    en <= not en;
    wait for 20 ns;
    en <= not en;
wait for clk_period*105;
--LINEA 17
        px <= x"3DED18D2";
    py <= x"3D52D037";
    pz <= x"BDD73EAB";

    en <= not en;

```

```

        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 18
        px <= x"3DEE45C3";
        py <= x"3D4F7553";
        pz <= x"BDD73EAB";
        en <= not en;
        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 19
        px <= x"3DEF722E";
        py <= x"3D4C1A55";
        pz <= x"BDD73EAB";
        en <= not en;
        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 20
        px <= x"3DF09F1F";
        py <= x"3D48BF72";
        pz <= x"BDD73EAB";
        en <= not en;
        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 21
        px <= x"3DF1CB8A";
        py <= x"3D456473";
        pz <= x"BDD73EAB";
        en <= not en;
        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 22

```

```

        px <= x"3DF2F87B";
    py <= x"3D420990";
    pz <= x"BDD73EAB";

    en <= not en;
    wait for 20 ns;
    en <= not en;
wait for clk_period*105;
--LINEA 23
        px <= x"3DF424E6";
    py <= x"3D3EAE91";
    pz <= x"BDD73EAB";

    en <= not en;
    wait for 20 ns;
    en <= not en;
wait for clk_period*105;
--LINEA 24
        px <= x"3DF551D7";
    py <= x"3D3B53AE";
    pz <= x"BDD73EAB";

    en <= not en;
    wait for 20 ns;
    en <= not en;
wait for clk_period*105;
--LINEA 25
        px <= x"3DF67EC8";
    py <= x"3D37F8B0";
    pz <= x"BDD73EAB";

    en <= not en;
    wait for 20 ns;
    en <= not en;
wait for clk_period*105;
--LINEA 26
        px <= x"3DF7AB32";
    py <= x"3D349DCC";
    pz <= x"BDD73EAB";

    en <= not en;

```

```

        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 27
        px <= x"3DF8D823";
        py <= x"3D3142CE";
        pz <= x"BDD73EAB";
        en <= not en;
        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 28
        px <= x"3DFA048E";
        py <= x"3D2DE7EA";
        pz <= x"BDD73EAB";
        en <= not en;
        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 29
        px <= x"3DFB317F";
        py <= x"3D2A8CEC";
        pz <= x"BDD73EAB";
        en <= not en;
        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
--LINEA 30
        px <= x"3DFC5DEA";
        py <= x"3D273209";
        pz <= x"BDD73EAB";
        en <= not en;
        wait for 20 ns;
        en <= not en;
wait for clk_period*105;
wait for clk_period*1000;

```

end process;  
END;

## **Anexo 08: Generalidades del dispositivo SPARTAN 3E FPGA**



**Module 1:  
Introduction and Ordering Information****DS312 (v4.1) July 19, 2013**

- Introduction
- Features
- Architectural Overview
- Package Marking
- Ordering Information

**Module 2:  
Functional Description****DS312 (v4.1) July 19, 2013**

- Input/Output Blocks (IOBs)
  - Overview
  - SelectIO™ Signal Standards
- Configurable Logic Block (CLB)
- Block RAM
- Dedicated Multipliers
- Digital Clock Manager (DCM)
- Clock Network
- Configuration
- Powering Spartan®-3E FPGAs
- Production Stepping

**Module 3:  
DC and Switching Characteristics****DS312 (v4.1) July 19, 2013**

- DC Electrical Characteristics
  - Absolute Maximum Ratings
  - Supply Voltage Specifications
  - Recommended Operating Conditions
  - DC Characteristics
- Switching Characteristics
  - I/O Timing
  - SLICE Timing
  - DCM Timing
  - Block RAM Timing
  - Multiplier Timing
  - Configuration and JTAG Timing

**Module 4:  
Pinout Descriptions****DS312 (v4.1) July 19, 2013**

- Pin Descriptions
- Package Overview
- Pinout Tables
- Footprint Diagrams

## Introduction

The Spartan®-3E family of Field-Programmable Gate Arrays (FPGAs) is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The five-member family offers densities ranging from 100,000 to 1.6 million system gates, as shown in [Table 1](#).

The Spartan-3E family builds on the success of the earlier Spartan-3 family by increasing the amount of logic per I/O, significantly reducing the cost per logic cell. New features improve system performance and reduce the cost of configuration. These Spartan-3E FPGA enhancements, combined with advanced 90 nm process technology, deliver more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry.

Because of their exceptionally low cost, Spartan-3E FPGAs are ideally suited to a wide range of consumer electronics applications, including broadband access, home networking, display/projection, and digital television equipment.

The Spartan-3E family is a superior alternative to mask programmed ASICs. FPGAs avoid the high initial cost, the lengthy development cycles, and the inherent inflexibility of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary, an impossibility with ASICs.

## Features

- Very low cost, high-performance logic solution for high-volume, consumer-oriented applications
- Proven advanced 90-nanometer process technology
- Multi-voltage, multi-standard SelectIO™ interface pins
  - Up to 376 I/O pins or 156 differential signal pairs

- LVC MOS, LVTTTL, HSTL, and SSTL single-ended signal standards
- 3.3V, 2.5V, 1.8V, 1.5V, and 1.2V signaling
- 622+ Mb/s data transfer rate per I/O
- True LVDS, RSDS, mini-LVDS, differential HSTL/SSTL differential I/O
- Enhanced Double Data Rate (DDR) support
- DDR SDRAM support up to 333 Mb/s
- Abundant, flexible logic resources
  - Densities up to 33,192 logic cells, including optional shift register or distributed RAM support
  - Efficient wide multiplexers, wide logic
  - Fast look-ahead carry logic
  - Enhanced 18 x 18 multipliers with optional pipeline
  - IEEE 1149.1/1532 JTAG programming/debug port
- Hierarchical SelectRAM™ memory architecture
  - Up to 648 Kbits of fast block RAM
  - Up to 231 Kbits of efficient distributed RAM
- Up to eight Digital Clock Managers (DCMs)
  - Clock skew elimination (delay locked loop)
  - Frequency synthesis, multiplication, division
  - High-resolution phase shifting
  - Wide frequency range (5 MHz to over 300 MHz)
- Eight global clocks plus eight additional clocks per each half of device, plus abundant low-skew routing
- Configuration interface to industry-standard PROMs
  - Low-cost, space-saving SPI serial Flash PROM
  - x8 or x8/x16 parallel NOR Flash PROM
  - Low-cost Xilinx® Platform Flash with JTAG
- Complete Xilinx ISE® and WebPACK™ software
- MicroBlaze™ and PicoBlaze™ embedded processor cores
- Fully compliant 32-/64-bit 33 MHz PCI support (66 MHz in some devices)
- Low-cost QFP and BGA packaging options
- Common footprints support easy density migration
- Pb-free packaging options
- [XA Automotive version](#) available

Table 1: Summary of Spartan-3E FPGA Attributes

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)				Distributed RAM bits <sup>(1)</sup>	Block RAM bits <sup>(1)</sup>	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs	Total Slices						
XC3S100E	100K	2,160	22	16	240	960	15K	72K	4	2	108	40
XC3S250E	250K	5,508	34	26	612	2,448	38K	216K	12	4	172	68
XC3S500E	500K	10,476	46	34	1,164	4,656	73K	360K	20	4	232	92
XC3S1200E	1200K	19,512	60	46	2,168	8,672	136K	504K	28	8	304	124
XC3S1600E	1600K	33,192	76	58	3,688	14,752	231K	648K	36	8	376	156

### Notes:

1. By convention, one Kb is equivalent to 1,024 bits.

© Copyright 2005–2013 Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE, Artix, Kintex, Zynq, Vivado, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. PCI and PCI-X are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.

## Architectural Overview

The Spartan-3E family architecture consists of five fundamental programmable functional elements:

- **Configurable Logic Blocks (CLBs)** contain flexible Look-Up Tables (LUTs) that implement logic plus storage elements used as flip-flops or latches. CLBs perform a wide variety of logical functions as well as store data.
- **Input/Output Blocks (IOBs)** control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. Supports a variety of signal standards, including four high-performance differential standards. Double Data-Rate (DDR) registers are included.
- **Block RAM** provides data storage in the form of 18-Kbit dual-port blocks.
- **Multiplier Blocks** accept two 18-bit binary numbers as inputs and calculate the product.

- **Digital Clock Manager (DCM) Blocks** provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase-shifting clock signals.

These elements are organized as shown in Figure 1. A ring of IOBs surrounds a regular array of CLBs. Each device has two columns of block RAM except for the XC3S100E, which has one column. Each RAM column consists of several 18-Kbit RAM blocks. Each block RAM is associated with a dedicated multiplier. The DCMs are positioned in the center with two at the top and two at the bottom of the device. The XC3S100E has only one DCM at the top and bottom, while the XC3S1200E and XC3S1600E add two DCMs in the middle of the left and right sides.

The Spartan-3E family features a rich network of traces that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.

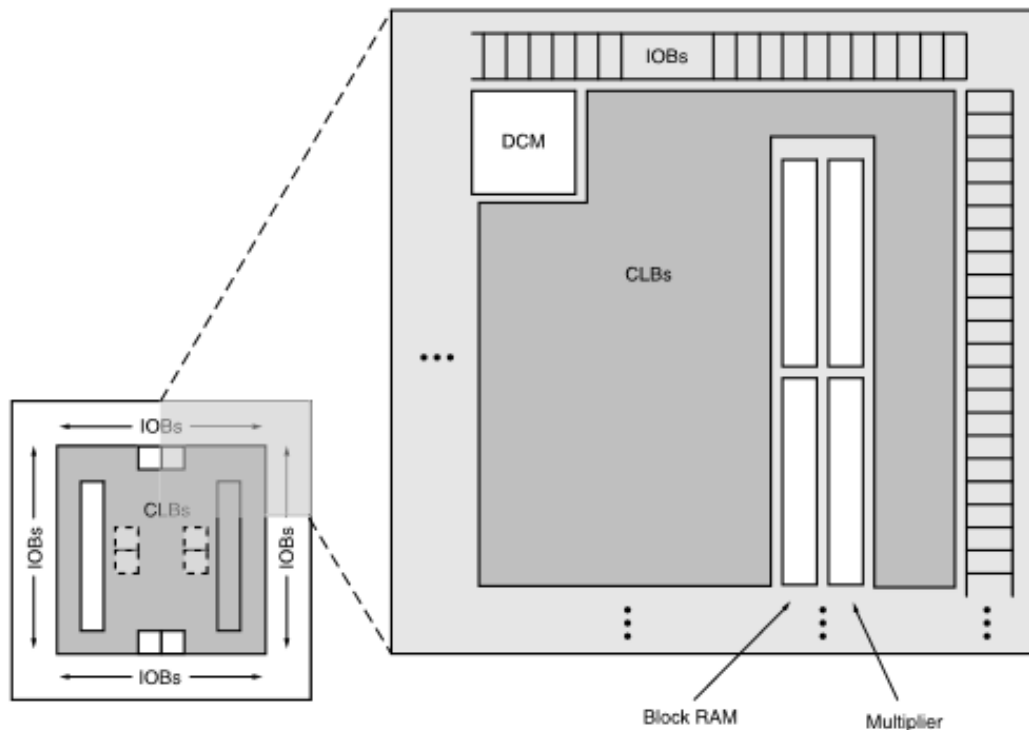


Figure 1: Spartan-3E Family Architecture

## Configurable Logic Block (CLB) and Slice Resources

For additional information, refer to the "Using Configurable Logic Blocks (CLBs)" chapter in [UG331](#).

### CLB Overview

The Configurable Logic Blocks (CLBs) constitute the main logic resource for implementing synchronous as well as combinatorial circuits. Each CLB contains four slices, and each slice contains two Look-Up Tables (LUTs) to implement logic and two dedicated storage elements that can be used as flip-flops or latches. The LUTs can be used as a 16x1 memory (RAM16) or as a 16-bit shift register

(SRL16), and additional multiplexers and carry logic simplify wide logic and arithmetic functions. Most general-purpose logic in a design is automatically mapped to the slice resources in the CLBs. Each CLB is identical, and the Spartan-3E family CLB structure is identical to that for the Spartan-3 family.

### CLB Array

The CLBs are arranged in a regular array of rows and columns as shown in [Figure 14](#).

Each density varies by the number of rows and columns of CLBs (see [Table 9](#)).

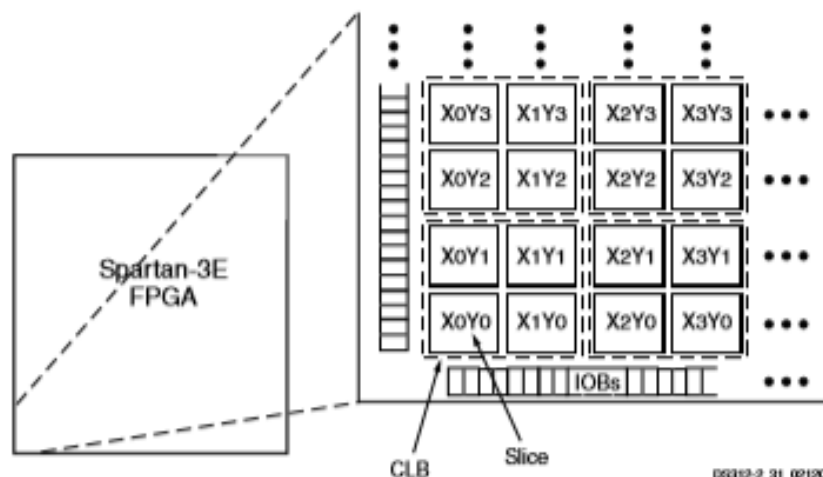


Figure 14: CLB Locations

DS312-2.91\_021206

Table 9: Spartan-3E CLB Resources

Device	CLB Rows	CLB Columns	CLB Total <sup>(1)</sup>	Slices	LUTs / Flip-Flops	Equivalent Logic Cells	RAM16 / SRL16	Distributed RAM Bits
XC3S100E	22	16	240	960	1,920	2,160	960	15,360
XC3S250E	34	26	612	2,448	4,896	5,508	2,448	39,168
XC3S500E	46	34	1,164	4,656	9,312	10,476	4,656	74,496
XC3S1200E	60	46	2,168	8,672	17,344	19,512	8,672	138,752
XC3S1600E	76	58	3,688	14,752	29,504	33,192	14,752	236,032

#### Notes:

- The number of CLBs is less than the multiple of the rows and columns because the block RAM/multiplier blocks and the DCMs are embedded in the array (see [Figure 1](#) in Module 1).

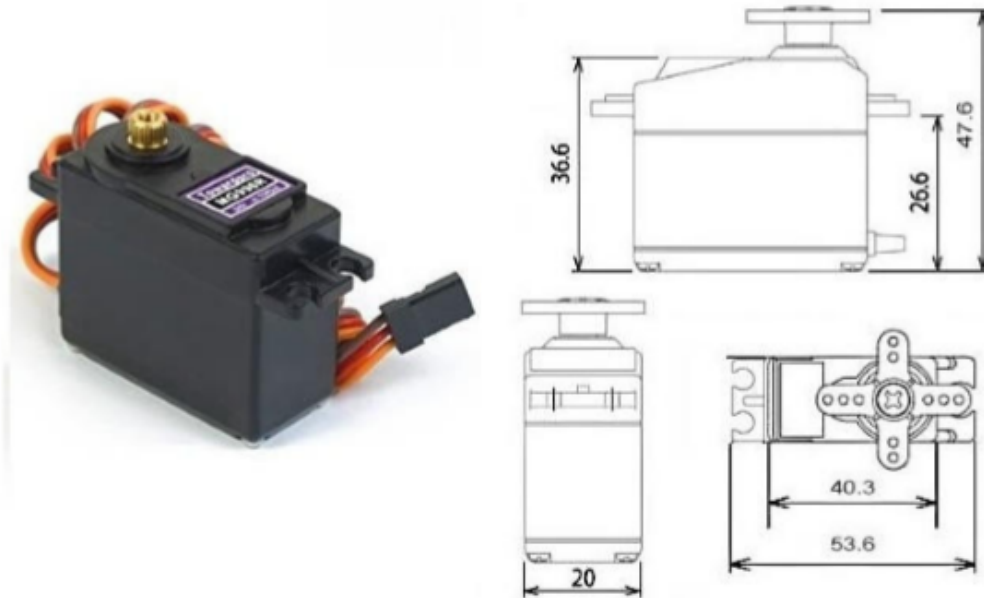
### Slices

Each CLB comprises four interconnected slices, as shown in [Figure 16](#). These slices are grouped in pairs. Each pair is organized as a column with an independent carry chain. The left pair supports both logic and memory functions and its slices are called SLICEM. The right pair supports logic only and its slices are called SLICEL. Therefore half the

LUTs support both logic and memory (including both RAM16 and SRL16 shift registers) while half support logic only, and the two types alternate throughout the array columns. The SLICEL reduces the size of the CLB and lowers the cost of the device, and can also provide a performance advantage over the SLICEM.



## MG996R High Torque Metal Gear Dual Ball Bearing Servo



This High-Torque MG996R Digital Servo features metal gearing resulting in extra high 10kg stalling torque in a tiny package. The MG996R is essentially an upgraded version of the famous MG995 servo, and features upgraded shock-proofing and a redesigned PCB and IC control system that make it much more accurate than its predecessor. The gearing and motor have also been upgraded to improve dead bandwidth and centering. The unit comes complete with 30cm wire and 3 pin 'S' type female header connector that fits most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum and Hitec.

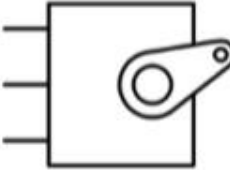
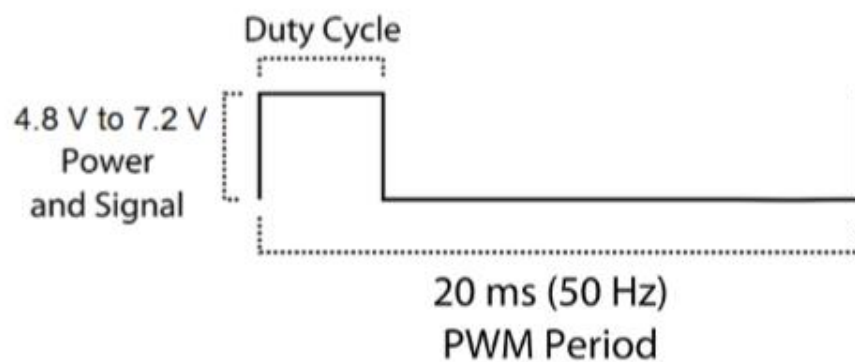
This high-torque standard servo can rotate approximately 120 degrees (60 in each direction). You can use any servo code, hardware or library to control these servos, so it's great for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. The MG996R Metal Gear Servo also comes with a selection of arms and hardware to get you set up nice and fast!

### Specifications

- Weight: 55 g
- Dimension: 40.7 x 19.7 x 42.9 mm approx.
- Stall torque: 9.4 kgf·cm (4.8 V), 11 kgf·cm (6 V)
- Operating speed: 0.17 s/60° (4.8 V), 0.14 s/60° (6 V)

- Operating voltage: 4.8 V a 7.2 V
- Running Current 500 mA – 900 mA (6V)
- Stall Current 2.5 A (6V)
- Dead band width: 5  $\mu$ s
- Stable and shock proof double ball bearing design
- Temperature range: 0 °C – 55 °C

PWM=Orange (  $\square$  )  
 Vcc = Red ( + )  
 Ground=Brown ( - )

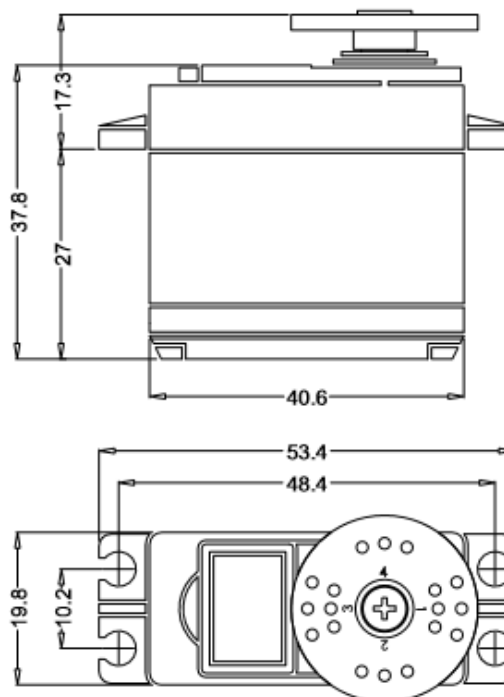



## Anexo 10: Generalidades del dispositivo HS-645MG Deluxe

# ANNOUNCED SPECIFICATION OF HS-645MG STANDARD DELUXE HIGH TORQUE SERVO

### 1. TECHNICAL VALUES

CONTROL SYSTEM	:+PULSE WIDTH CONTROL 1500usec NEUTRAL	
OPERATING VOLTAGE RANGE	:4.8V TO 6.0V	
OPERATING TEMPERATURE RANGE	:-20 TO +60° C	
TEST VOLTAGE	:AT 4.8V	:AT 6.0V
OPERATING SPEED	:0.24sec/60° AT NO LOAD	:0.2sec/60° AT NO LOAD
STALL TORQUE	:7.7kg.cm(106.93oz.in)	:9.6kg.cm(133.31oz.in)
OPERATING ANGLE	:45°/ONE SIDE PULSE TRAVELING 400usec	
DIRECTION	:CLOCK WISE/PULSE TRAVELING 1500 TO 1900usec	
IDLE CURRENT	:8.8mA	:9.1mA
RUNNING CURRENT	:350mA	:450mA
DEAD BAND WIDTH	:8usec	
CONNECTOR WIRE LENGTH	:300mm(11.81in)	
DIMENSIONS	:40.6x19.8x37.8mm(1.59x0.77x1.48in)	
WEIGHT	:55.2g(1.94oz)	



### 2. FEATURES

3-POLE FERRITE MOTOR  
DUAL BALL BEARING  
LONG LIFE POTENTIOMETER  
3-METAL GEARS & 1-RESIN METAL GEAR  
HYBRID I.C

### 3. APPLICATIONS

AIRCRAFT TO 1/4 SCALE  
30 TO 60 SIZE HELICOPTERS  
STEERING AND THROTTLE FOR 1/10TH & 1/8TH ON-ROAD AND OFF-ROAD VEHICLES

## Anexo 11: Generalidades del dispositivo AX-12 Dynamixel

ROBOTIS e-Manual v1.10.00

### AX-12/ AX-12+/ AX-12A

---

#### Part Photo

---



※ AX-12+ is the improved version of existing AX-12: the design of circuit, material, and wheel gear are specially improved.

※ AX-12A is a new version of the AX-12+ with the same performance but more advanced external design. Only the AX-12A is now being sold.

#### H/W Specification

---

- Weight : 53.5g (AX-12/AX-12+), 54.6g (AX-12A)
- Dimension : 32mm × 50mm × 40mm
- Resolution : 0.29°
- Gear Reduction Ratio : 254 : 1
- Stall Torque : 1.5N.m (at 12.0V, 1.5A)
- No load speed : 59rpm (at 12V)
- Running Degree
  - 0° ~ 300°
  - Endless Turn
- Running Temperature : -5°C ~ +70°C
- Voltage : 9 ~ 12V (Recommended Voltage 11.1V)
- Command Signal : Digital Packet
- Protocol Type : Half duplex Asynchronous Serial Communication (8bit, 1stop, No Parity)
- Link (Physical) : TTL Level Multi Drop (daisy chain type Connector)
- ID : 254 ID (0~253)
- Communication Speed : 7343bps ~ 1 Mbps
- Feedback : Position, Temperature, Load, Input Voltage, etc.
- Material : Engineering Plastic

#### Control Table

---

Control Table consists of data regarding the current status and operation, which exists inside of Dynamixel. The user can control Dynamixel by changing data of Control Table via Instruction Packet.

##### EEPROM and RAM

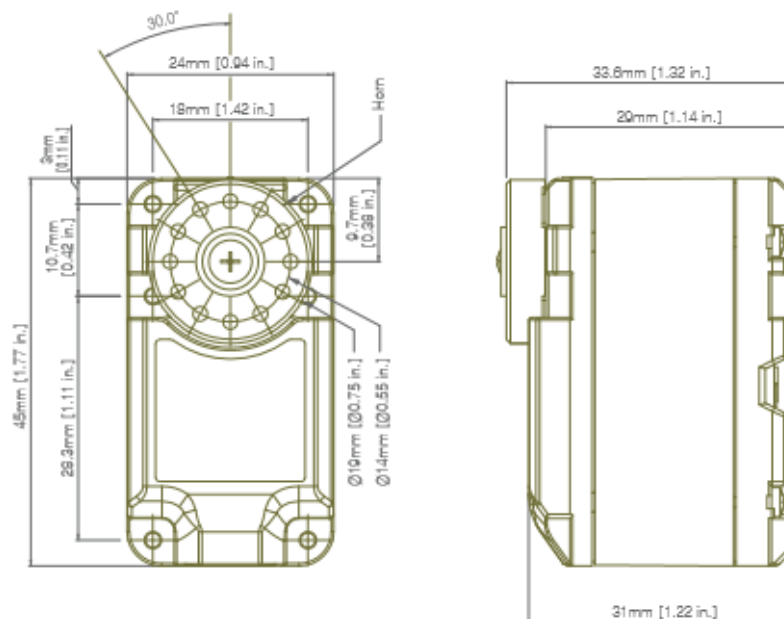
Data in RAM area is reset to the initial value whenever the power is turned on while data in EEPROM area is kept once the



## Anexo 12: Generalidades del dispositivo DRS-0101 HerkuleX

### 2-3. 제품 사양

Dimension / Weight	45mm(W) x 24.0mm(D) x 31mm(H) / 45g [1.59 oz] 45mm(W) x 24.0mm(D) x 31mm(H) / 60g [2.12 oz] (DRS-0201) [1.77 in.(W) x 0.94 in.(D) x 1.22 in.(H)]
Reduction Ratio Gear Material	1 : 266 Super Engineering Plastic, Heavy Duty Metal (DRS-0201)
Input Voltage Rated Current Motor	7~12VDC(Optimized 7.4V) 450mA @ 7.4V : 1.7kgf.cm, 670mA @ 7.4V : 2.2kgf.cm (DRS-0201) Carbon Brush Cored DC, Metal Brush Coreless DC (DRS-0201)
Stall Torque / Maximum Speed	12kgf.cm [166.8 ozf.in.] / 0.166s/60° @7.4V 24kgf.cm [333.6 ozf.in.] / 0.147s/60° @7.4V (DRS-0201)
Resolution	0.325°
Operating Angle Temperature	320°, Continuous Rotation 0 ~ 85°C [32°F~185°F]
Communication Link ID, Maximum Baud Rate	Full Duplex Asynchronous Serial(TTL Level), Binary Packet, Multi Drop 0 ~ 253, 254(Broadcast only) 0.67Mbps
Feedback	Position, Speed, Temperature, Load, Voltage etc.
Control Algorithm	PID, Feedforward, Trapezoidal Velocity Profile, Velocity Override, Torque Saturator & Offset, Overload Protection, Neutral Calibration, Dead Zone 54 Selectable Setting Parameters(* Servo Manager Kit Required)



※ 커넥터 사양은 page 51, 52를 참고 하십시오.